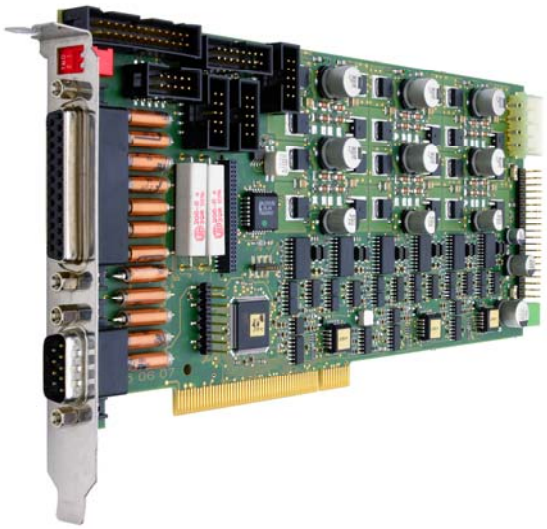
Feinpositioniersysteme

LSTEP / ECO-STEP

LSTEP / PCI



LANG GMBH & CO. KG
Dillstraße 4
D-35625 Hüttenberg
Tel. 06403/7009-0
Telefax 06403/7009-40

Sehr geehrter Kunde!

Vielen Dank, daß Sie sich für eine Steuerung aus unserem Hause entschieden haben!

Mit dem Gerät haben Sie eine Positioniersteuerung gewählt, die bei einem geringen Platzbedarf anspruchsvolle Positionieraufgaben automatisiert ausführt. Die hohe Präzision der Steuerung eröffnet Ihnen weite Anwendungsmöglichkeiten. Die Schrittauflösung von bis zu 50.000 (400.000) Schritten pro Motorumdrehung, bei einem zweihundertschrittigen Motor und 2000 Schritte pro Vollschritt bei Linearschrittmotoren, bieten Auflösungen im sub-µm-Bereich. Weiter bietet der „closed loop" Betrieb in Verbindung mit einer hochauflösenden Geberauswertung bei optischen und magnetischen Meßsystemen eine sehr hohe Positioniergenauigkeit.
Die vielen Zusatzfunktionen, wie z.B. Snapshot, Triggerout, Takt- und Drehrichtungs- Ein- und Ausgänge machen die Steuerung zu einem idealen Partner für viele Applikationen.

Vor der Inbetriebnahme Ihrer Steuerung bitten wir Sie, diese Anleitung sorgfältig und in Ruhe zu lesen.

Achten Sie insbesondere auf die Sicherheitshinweise!

Inhaltliche Änderungen behalten wir uns vor. Wir haften nicht für etwaige Fehler in dieser Dokumentation. Bedingt durch ständige technische Weiterentwicklungen unserer Produkte können mitunter leichte Abweichungen zwischen den Beschreibungen in dieser Dokumentation und Ihrer Maschine bestehen. Eine Haftung für mittelbare Schäden, die im Zusammenhang mit der Lieferung oder dem Gebrauch dieser Dokumentation entstehen, ist ausgeschlossen, soweit dies gesetzlich zulässig ist.

Schutzvermerk nach DIN 34

Inhaltsverzeichnis

Seite

Seite

Seite

Seite

1 Sicherheitshinweise

1.1 Grundsätzliche Hinweise

- Instandsetzungs- und Reparaturarbeiten dürfen nur von Fachpersonal durchgeführt werden, das besonders geschult und mit der Steuerung bestens vertraut ist!

- Vor dem Öffnen des Gerätes ist der Netzstecker zu ziehen!

- Die Leistungsaufnahme bei der LSTEP-2x/2 kann kurzfristig auf 200 VA ansteigen wenn alle drei Achsen mit 2,5 A und maximaler Geschwindigket betrieben werden. Diese Leistungsaufnahme ist jedoch für den Dauerbetrieb nicht zulässig, da auch die LSTEP-2x/2 ohne zusätzliche Kühlung (Lüfter) arbeitet. Die mittlere Leistungsaufnahme darf hierbei 100 VA nicht überschreiten!

- Bei der Steuerungsausführung LSTEP 22/2 (3,75) ist die Stromaufnahme im Stand auf mindestens 75% abzusenken!

- Es dürfen nur Geräte angeschlossen werden die von uns spezifiziert sind.
  Zuwiderhandlungen können zu Zerstörung der Steuerung bzw. des angeschlossenen Gerätes führen!

- Der Netzschalter der Steuerung, oder die Steckdose an der die Steuerung angeschlossen ist muß immer erreichbar sein, damit Jederzeit die Steuerung allpolig vom Netz getrennt werden kann!
- **In eingeschaltetem Zustand dürfen keine Kabel gesteckt oder abgezogen werden!**

1.2 Hinweise zur Inbetriebnahme

- Schnittstellenbelegung der LSTEP.
  Der Commander (Terminal für LSTEP und MCL) wird über den Schnittstellenanschluß mit einer geregelten Gleichspannung von +5V oder +12V versorgt. Die Spannung wird bei der LSTEP über die Steckbrücke J1 (hinter ST2) auf PIN 1 von ST2 gelegt, sofern ein Terminal mit der Steuerung geliefert wird.
  Verwenden Sie deshalb bitte immer das original Schnittstellenkabel.
  Hinweis für ECO-STEP: An Stelle eines Jumpers ist eine Drahtbrücke vorhanden.

- Einstellung der Netzspannung.
  Die LSTEP kann entweder an 100V - 120V oder an 200V - 240V betrieben werden. Die jeweilige Spannung wird über einen steckbaren Spannungswähler mit Sicherungsträger am Netzeingang eingestellt. Achten Sie unbedingt darauf, daß das Gerät immer an der eingestellten Spannung betrieben wird. Wenn der Spannungswähler auf 100V - 120V eingestellt ist und die LSTEP an 200V - 240V angeschlossen wird, kann es zur **Zerstörung der Steuerelektronik** kommen. Auf jeden Fall brennt die Netzeingangssicherung durch!
  Hinweis für ECO-STEP: Die ECO-STEP wird von einem externen Tischnetzteil mit Weitbereichseingang 100-240V versorgt.

- Lüftungsschlitze im Gehäuse.
  Da es notwendig ist die Leistungselektronik der Steuerung duch Lüftung zu kühlen, sind im Gehäuse Lüftungsschlitze eingearbeitet. Es ist daher unbedingt darauf zu achten, daß keine Späne, Flüssigkeiten oder andere elektrisch leitende Substanzen in das Innere des Gehäuses gelangen. Dies gilt nur für die Steuerung LSTEP-3x/2 (Phasenstrom bis 5A).

- Schutz der angeschlossenen Mechanik
  Nach dem Einschalten der Steuerung sollte immer ein Ausmessen des Verfahrbereichs mit den Kommandos ”Kalibrieren” und ”Tischhub messen” erfolgen. Dadurch ist die Steuerung in der Lage, Verfahrbewegungen, die den maximalen Verfahrbereich überschreiten, zu erkennen und zu vermeiden (vgl. Kapitel 5.1, Kapitel 5.2).
  Werden Verfahrbereichsgrenzen gesetzt, so fährt die Achse nur bis zu den eingestellten Grenzen. Dies ist notwendig, wenn man nur einen Endschalter pro Achse zur Verfügung hat.

2 Funktionsbeschreibung

Mit der Schrittmotorsteuerung LSTEP werden Koordinatentische für z.B. Mikroskope oder Fertigungsabläufe mit Auflösungen bis 0,0001 mm betrieben. Die Steuerung zeichnet sich durch hohe Laufruhe der Motoren aus. Durch das dynamische Mikrostep - Antriebs - Prinzip lassen sich trotz einer hohen Auflösung von 50.000 (400.000) Mikroschritten je Motorumdrehung hohe Drehzahlen von bis zu 40 U/sec (7,5 U/sec) bei einem 200 schrittigen Motor erreichen. Für Linear-Schrittmotore werden eigene Motortabellen mit 2000 Mikroschritten pro Vollschritt eingesetzt, d. h. 8000 Mikroschritte pro Zahnteilung.
Die Steuerung arbeitet mit linearer Interpolation (alle Achsen erreichen gleichzeitig die Zielposition) und automatischer frei programmierbarer Rampengenerierung (Begrenzung der Beschleunigung beim Starten und Stoppen). Die LSTEP kann alleinstehend oder über einen PC gesteuert betrieben werden. Eine Positionsanzeige (Option) an der Frontplatte sowie ein "Joy-Stick" ergänzen dabei das Gerät. Für die LSTEP wurde ein neuer Befehlssatz entwickel, welcher wesentlich mehr Funktionalität bietet. Auch der Register-Befehlssatz der seit Jahren erfolgreich eingesetzten Steuerung "MCL" ist weiterhin verfügbar.

Für einen sauberen Ablauf und eine sichere Positionierung sollten Motoren mit einem Schrittwinkelfehler < ± 3% eingesetzt werden. Um die maximale Drehzahl zu erreichen sollten möglichst niederohmige Motoren mit geringer Induktivität eingesetzt werden.
Zur Vermeidung einer unnötigen Erwärmung der Motoren senkt die LSTEP in jeder Betriebspause (auch bei Joy-Stick - Betrieb) den Motorstrom auf den eingestellten Ruhestrom ab (vgl. Kapitel 6.7).

2.1 RS232 Schnittstelle

Als Standard Schnittstelle zum übergeordneten PC wird eine serielle Schnittstelle nach RS232 Spezifikation mit folgenden Standardeinstellungen verwendet:
- 9600 Baud, 11 Bit - Rahmen, 1 Start-, 8 Daten-, kein Paritäts-, 2 StopBit
Für einen problemlosen Betrieb benötigt die LSTEP einen PC-seitigen RS 232 - Anschluß mit folgenden aufgelegten Signalen:

RxD:	Empfängerleitung der LSTEP	(Sendeleitung des Rechners)
TxD:	Sendeleitung der LSTEP	(Empfängerleitung des Rechners)
RTS:	Sendefreigabe von der LSTEP	
CTS:	Sendefreigabe des PC	
GND:	Signalmasse	

Mit Einschränkungen ist ein Betrieb ohne RTS - Leitung möglich.

2.1.1 Betrieb ohne Steuerrechner

Einfache Bewegungen sind mit der LSTEP ohne Steuerrechner durchführbar. Dazu wird der Schalter "Joy-Stick" auf "manuel" gestellt. Nun können mit dem Joy-Stick beliebige Positionen angefahren werden. Bei Steuerungen mit LC-Display wird die augenblickliche Absolutposition dabei ständig angezeigt. Weiterhin können mit Hilfe der Schalter "CLEAR" die Achsen einzeln auf Null gesetzt werden.

2.2 Bedienelemente

Die Anzeige (nur bei Geräten mit Anzeige) und sämtliche Bedienungselemente mit Ausnahme des Netzschalters befinden sich an der Frontplatte.

Bedienelement	Bedeutung
CLEAR X/Y/Z (optional)	Schalter zum Nullsetzen der Anzeige (getrennt für X- Y-und Z- Position). Mit der Anzeige werden auch die Positionsregister gelöscht.
SPEED 1..10 (optional)	Potentiometer zum Ändern der Motordrehzahl bei Fahren mit externem Takt. Der Softwareseitig eingestellte Wert kann von 0 bis 100% eingestellt werden. Der vor Beginn eines Vektors eingestellte Potentiometerwert gilt für das Verfahren des gesamten Vektors und kann nicht während des Verfahrens verändert werden. Ist der Joy-Stick aktiv, kann die Verfahrgeschwindigkeit mit dem Potentiometer verändert werden. Hinweis: Besonders für Joystick-Handbetrieb interessant.
JOYSTICK MAN / AUTO	Joy-Stick Wahlschalter MAN = Handbetrieb (es können keine Move-Befehle ausgeführt werden) AUTO = Automatikbetrieb mit den entsprechenden Befehlen
RESET (optional)	Bei Bedienung des Reset-Schalters nach oben wird die Steuerung in Grundstellung (wie nach aus- und einschalten) gebracht.
ON	Betriebsanzeige der LSTEP
LCD Anzeige (optional)	LCD-Anzeige mit 4*16 Zeichen zum Anzeigen der Betriebsart und Absolutposition. Angezeigt werden Positionen mit dem Wertebereich $-99.999.999,9 \leq P \leq +99.999.999,9$.

Tabelle 1: Bedienelemente auf der Frontplatte der LSTEP

Joystick Schalter in Stellung "AUTO"		
	EMPFANGSBEREIT (Ready to receive)	LSTEP wartet auf Befehle über RS232 Schnittstelle
	POSITION FAHREN (go to position)	LSTEP fährt eine Position an
	RELATIVE GERADE (relative straight line)	LSTEP fährt eine relative Gerade
	CALIBRIEREN (Calibration)	LSTEP fährt in Nullposition
	TISCHSCHLAENGE	LSTEP fährt in Endposition
	JOYSTICK AUTO	LSTEP fährt mit Joystick Bedienung
Joystick Schalter in Stellung H		
	JOYSTICK MAN	LSTEP fährt mit Joystick Bedienung

Tabelle 2: Betriebsarten der LSTEP

3 Inbetriebnahme

ACHTUNG:	Die Lüftung an der Rückwand und im Bodenblech des Gerätes (LSTEP-3x/2) darf nicht abgedeckt werden!

3.1 Anschlüsse

- Motoren über die mitgelieferten Kabel anschließen.
- Inkrementale Meßsysteme anschließen (wenn vorhanden).
- Joy-Stick anschließen und mit den Schiebern verriegeln.
- Rechner oder Commander über Interface-Kabel anschließen.
- Netz anschließen.

3.2 Anschlußdaten für Ein/Ausgänge–Multifunktionsport *(nicht bei ECO-STEP)*

Für die Ein/Ausgänge müssen folgende Anschlußwerte eingehalten werden

- digitale Eingänge (z.B. Takt Vor/Rück, Momententrigger)

Signalpegel:	TTL; max. Eingangsstrom ±5mA
vorhandene Eingangsbeschaltung	RC-Tiefpaß mit 470 Ω /220pF, 4.7Ω Pull-Up auf +5V

- digitale Ausgänge (Trigger-Out) — TTL-Pegel mit ±1,6 mA

3.3 Anschluß inkrementaler Meßsysteme *(nicht bei ECO-STEP)*

An die Steuerung können inkrementale Dreh- oder Lineargebersysteme zur Erkennung bzw. Vermeidung eines Schrittversatzes angeschlossen werden. Somit wird ein closed-loop Betrieb ermöglicht. Die Einsatzmöglichkeiten sind hierbei nicht auf optische Meßsysteme beschränkt. Z.B. können auch induktive oder magnetoresisitve Systeme ausgewertet werden, sofern deren Ausgangssignale die spezifizierten Grenzwerte einhalten. Das optionale Geberinterface ermöglicht den Anschluß von Gebersystemen mit sinusförmigen Ausgangssignalen.

Hierzu bestehen die folgenden zwei alternativen:

1. sinusförmige Spannungssignale 1V $_{SS.}$
2. Magnetisches Längenmeßsystem.

Aufgrund der begrenzten Datenbreite von Mikrocontrollern führen nicht alle Kombinationen von Spindelsteigungswerten und Geberteilungsperioden zu korrekten Ergebnissen bei der Positionsberechnung. Einige mögliche Kombinationen von Spindelsteigung und Periodenteilung linearer Meßsysteme sind in untenstehender Tabelle dokumentiert. Ein (X) bedeutet, daß die angegebene Kombination ohne Einschränkung verwendet werden kann.

	Geberteilung in mm						
Spindelstei-gung in mm	1,00 mm	0,50 mm	0,10 mm	0,020 mm	0,0080 mm	0,0040 mm	0,0001 mm
0,40 mm	X	X	X	X	X	X	X
0,50 mm	X	X	X	X	X	X	X
1,00 mm	X	X	X	X	X	X	X
2,00 mm	X	X	X	X	X	X	X
3,00 mm							
4,00 mm	X	X	X	X	X	X	X
5,00 mm	X	X	X	X	X	X	X
8,00 mm	X	X	X	X	X	X	X
10,00 mm	X	X	X	X	X	X	X
15,00 mm							
20,00 mm	X	X	X	X	X	X	
25,00 mm	X	X	X	X	X	X	
30,00 mm							
35,00 mm							
50,00 mm	X	X	X	X	X	X	
100,00 mm	X	X	X	X	X	X	

Tabelle: zulässige Geberteilungen (X) in Abhängigkeit der gewählten Spindelsteigung

Für die Fälle, die in der Tabelle nicht erfasst sind, kann nachfolgende Gleichung herangezogen werden.

$$geberfaktor = \frac{4 \cdot 10^5 \cdot t_p}{h}$$

$$h : Spindelsteigung\ in\ mm$$

$$t_p : Geberteilung\ in\ mm$$

Ergibt sich in Abhängigkeit der gewählten Spindelsteigung und der Periodenteilung des Meßsystems für *geberfaktor* ein ganzzahliger Wert ohne Nachkommastellen, so ist die gewählte Kombination ohne Einschränkung nutzbar.
In allen anderen Fällen wenden Sie sich bitte an den Hersteller der Steuerung.

3.4 Funktionstest

- Gerät einschalten
  Nach jedem Einschalten führt die LSTEP selbsttätig eine Kalibrierung des angeschlossenen Joystick durch, die ca. 5s in Anspruch nimmt. Um eine korrekte Kalibrierung zu gewährleisten, darf der Joystick während dieser Zeit nicht ausgelenkt werden.
- Schalter "Joy-Stick" auf "MAN"
- Joy-Stick in allen Richtungen auslenken: Die Motoren laufen entsprechend der Auslenkung. Wenn keine Reaktion erfolgt, dann Motor- und Joy-Stick - Anschlüsse prüfen. Sind die Anschlüsse in Ordnung, dann sollte das Gerät auf versteckte Transportschäden untersucht werden.
- Schalter "Joy-Stick" auf "AUTO"
- Funktionsaufruf (siehe Befehlssatz)

3.5 Lösung von Problemen bei der Inbetriebnahme der RS232 Verbindung

- LSTEP antwortet nicht über RS 232:
  - Anschlussbelegung und Anschlusskabel zum Steuerrechner prüfen
  - Schnittstellenbedingungen (OPEN-Befehl) am Steuerrechner prüfen

- Einzelne Bytes von Meldungen der LSTEP gehen verloren:
  - Am Steuerrechner steht keine CTS - Leitung zur Verfügung (RTS - Leitung der LSTEP wird nicht geprüft): Immer wenn die LSTEP arbeitet und keine Daten empfangen kann, wird die Schnittstelle über RTS gesperrt. Eine Synchronisierung des Rechners und der LSTEP erfolgt auch ohne Kontrolle der RTS-Leitung, wenn der Rechner wie in den Beispielen dieser Anleitung beschrieben auf die Statusmeldungen der LSTEP wartet. Probleme können sich jedoch bei den Funktionen "Auflösung und Spindelsteigung einstellen" ergeben, wenn über den Befehl „Autostatus“ kein sicheres Protokoll eingestellt wurde (siehe Befehlssatz). Hier muß der Rechner z.B. mit Hilfe von Schleifen verzögert werden, damit Daten oder Kommandos nicht verloren gehen. Die typische Verzögerungszeit beträgt ca. 20 msec.

3.6 Firmware update

Die Steuerung kann sehr einfach mit Programm Updates auf den neuesten Stand gebracht werden. Bitte beachten Sie hierzu, in Abhängigkeit der eingesetzten Steuerung, die nachfolgend beschriebene Vorgehensweise:

- Schließen Sie eine der seriellen Schnittstellen (COM) Ihres PC an die serielle Schnittstelle auf der Rückwand der Steuerung (LSTEP + ECO-STEP), oder an die LSTEP-PCI ST2 auf 9pol DSub
- Beenden Sie alle Programme die auf die gleiche Schnittstelle zugreifen.
- Kopieren Sie das selbstentpackende Programm „LFlash.exe“ auf Ihren PC und entpacken Sie es.
- Kopieren Sie das neue Steuerungs-Programm „*.ihx“ oder „*.hex“ auf Ihren PC
- Starten Sie „Flash.exe“ unter Windows
- Wählen Sie die Schnittstelle des PC aus, die Sie mit der Steuerung verbunden haben,
- Wählen Sie den Gerätetyp aus.(LSTEP; ECO-STEP)
- Der Dip-Schalter "1" auf der Rückwand muß auf ON gestellt und dann die Steuerung eingeschaltet werden. Bei der PCI-Karte machen Sie nach dem Einschalten von Dip-Schalter "1" mit Dip-Schalter "2" einen Reset (Ein-und Ausschalten) "
- Klicken Sie auf das Kästchen **Update** und bestätigen Sie mit "**Ja**". Das alte Programm in der Steuerung wird gelöscht. (Je nach Programm-Version, müssen alle oder nur die Bänke 0-2 im Flash gelöscht werden.
- Wählen Sie den Dateityp aus (ihx od. hex File).
- Laden Sie das neue Steuerungs-Programm.

➔ Die Firmware wird nun zur Steuerung übertragen.

- Ist der Programmiervorgang beendet, muß der Dip-Schalter "1" wieder in seine Ausgangsposition gebracht werden.

Nach einem erneuten Betätigen des Reset Tasters oder dem Ein.- und Ausschalten der Steuerung arbeitet die Steuerung mit der neuen Firmware.

3.6.1 Firmware update mit dem neuen Flashtool ab der Version 3.0.0.0

So führen Sie ein Update durch:

1. Wählen Sie Serielleschnittstelle aus.

2. Stellen Sie den DIP Schalter 1 an der Steuerung auf ON

3. Schalten Sie nun die Steuerung ein oder betätigen Sie den Resetschalter wenn die Steuerung noch eingeschaltet ist.

4. Starten Sie das Update. Dazu klicken Sie auf den Butten mit der Aufschrift "Update". Das Programm stellt nun automatisch eine Verbindung zur Steuerung her und löscht das Flash. Danach werden Sie aufgefordert die neue Steuerungs- software auszuwählen. Nachdem Sie die Datei ausgewählt haben wird diese im Flash gespeichert.

5. Nach erfolgreichem Update stellen Sie den DIP Schalter 1 wieder auf 'OFF'

6. Betätigen Sie nun den Resetschalter oder schalten Sie die Steuerung aus und ein. Damit starten Sie die Steuerung mit der neuen Software.

4 Der Befehlssatz der LSTEP-Steuerungen

Zur besseren Übersicht werden alle Befehle und Parameter die an die Steuerung geschickt werden, sowie alle Rückmeldungen der Steuerung als ASCII-Charakter übertragen. Dies hat den Vorteil, daß die Befehle einerseits über ein normales Terminal manuell vorgegeben werden können. Zum anderen erleichtern diese Klartextbefehle die Fehlersuche, wenn die Befehle über ein kundenspezifisches Programm vorgegeben werden.

Kommandos oder Parameter, die an die Steuerung übertragen werden, beginnen mit einem Ausrufezeichen „!“. Abfragen werden durch ein Fragezeichen „?“ gekennzeichnet. Beispielsweise bedeuten:

!cal	*Kalibrieren*
?status	*Auslesen des Status*

Hinweis:	Bei Befehlen, die nur ein Schreiben oder nur ein Lesen zulassen, können die Zeichen „!“ bzw. „?“ entfallen.

Manche Befehle, z.B. die Vorgabe von Verfahrbewegungen, erfordern die Übergabe von Parametern. Diese werden im Anschluß an den eigentlichen Befehl übertragen. Zwischen Kommandotext und Parametern sowie zwischen verschiedenen Parametern müssen Leerzeichen als Trennung eingefügt und übertragen werden.

moa 45 13 20	*Verfahre x, y und z an die Positionen 45, 13 und 20*

Jeder Befehl muß mit einem Carriage Return (CR) abgeschlossen werden. Dieses Zeichen wird im ASCII-Zeichensatz folgendermaßen dargestellt:

Symb. Name	dez. Wert	hex. Wert	bin. Wert
CR	13	0xD	00001101

4.1 Kurzbeschreibung der Lstep-Befehle

Befehl	Beispiel	Bemerkung	Kap.4 Seite
Schnittstelle			
baud	(?) !baud 9600	Baudrate auf 9600 einstellen	13
cts	(?) !cts 0	Die CTS-Auswertung ist deaktiviert	13
intcom	(?) !intcom 1	Kommunikation über DPRAM Interruptgesteuert	13
Steuerungs-Informationen			
ver	?ver	Versionsnummer auslesen	10
iver	?iver	Ergänzung zur aktuellen Versionsnummer	10
det	?det	detallierte Versionsnummer auslesen	11
readsn	?readsn	Seriennummer der Steuerung auslesen	12

Befehl	Beispiel	Bemerkung	Kap.4 Seite
Einstellungen			
ipreter	(?) !ipreter 0; 1; oder 2	Umschalten des Befehlssatzes	14
xycomp	(?) !xycomp (1-6)	für Antriebe die sich beeinflussen	40
dim	(?) !dim 1	Einstellen der Einheit in µm	23
pitch	(?) !pitch 1 1 1 (y 4)	Einstellen der Spindelsteigung X Y Z oder nur Y	24
gear	(?) !gear	Getriebefaktor	24
accel	(?) !accel 1 1 1 (x 1)	Einstellen der Beschleunigung X Y Z oder nur X	25
vel	(?) !vel 10 10 10 (x 20)	Einstellen der Geschwindigkeit X Y Z oder nur X	25
velfac	(?) !velfac (1-100)	Reduzierung der eingestellten Geschwindigkeit	26
pot	(?) !pot 1(0)	Speedpoti Ein.-/Ausschalten	42
cur	(?) !cur 1 2 2.5	Motorstromeinstellung: X=1A Y=2A Z=2,5A	27
maxcur	?maxcur	alle max.Ströme werden angezeigt (=Konfiguration)	26
reduction	(?) !reduction 0.5 0.5 0.5	Stromabsenkung auf 50% in allen Achsen	27
curdelay	(?) !curdelay 1000	Verzögerung für die Stromabsenkung (0 - 10000 ms)	28
opfl	(?) !opfl 20 20 20 20	Ab 20 Umdr./sec. wird mit maxstrom gefahren	21
axis	(?) !axis1 0 1 (y 1)	Achsen Ein.- und Ausschalten	28
axisdir	(?) !axisdir 0 1 0	Motordrehrichtung für Y-Achse gedreht	29
caliboffset	(?) !caliboffset 1 1 1 1	Der Nullpunkt wird um 1mm verschoben bei Dim 2	35
rmoffset	(?) !rmoffset 1 1 1 1	Die Endposition wird um 1mm verschoben bei Dim 2	35
caldir	(?) !caldir z 1	Die Z-Achse wird in positive Richtung Kalibriert	36
calbspeed	!calbspeed 10	bei "cal"+"rm" wird mit 0,1U/s aus den Endschaltern gefahren (5...100)	37
calrefspeed	!calrefspeed 0-100	Geschwindigkeit mit der beim Kalibrieren die Referenzmarke gesucht wird	37
Save	save	die aktuellen Parameter werden ins Flash gebrannt	15
savejoyonoff	(?) !savejoyonoff 1 (0)	nach anschließendem Save und Reset, ist bei der LSTEP-PCI nach dem Einschalten des PCs der Joystick aktiv.	49
saveipreter	(?) !saveipreter 0	nach anschließendem Save und Reset, ist die LSTEP fest auf den Registerbefehlssatz umgestellt	15
Reset	Reset	Die Software wird in den Startzustand versetzt	12
pa	pa 1	Poweramplifier 1 (Endstufe eingeschaltet) 0=ausgeschaltet (nur bei der LSTEP-44)	12
vlevel	(?) !vlevel 1 0.8 !vlevel 2 1.2	Ausblenden von Geschwindigkeiten, bei denen Resonanzen auftreten.	21
mtpatch	(?) !mtpatch 1	Die Korrekturtabelle wurde aktiviert	22
joyfilter	(?) !joyfilter 1	Filterung und Hysterese im Joystickbetrieb aktiviert	22
stoppol	(?) !stoppol 0 oder 1	Der Stopeingang (MFP) ist low oder high aktiv	33
stopaccel	(?)!stopaccel 2	Die Bremsbeschleunigung, wenn der Stopeingang aktiv wird, ist 2m/s²	33

Befehl	Beispiel	Bemerkung	Kap.4 Seite
Statusabfragen			
autostatus	(?) !autostatus 0 (0-4)	Einstellung der Rückmeldung von der Steuerung	16
Status	?status	Liefert den aktuellen Zustand der Steuerung	16
Statusaxis	?statusaxis	aktuellen Zustand der einzelnen Achsen (@,M,J,C,S,A,D,-,)	17
err	?err	liefert die aktuelle Fehlernummer	17
statuslimit	?statuslimit	A=Kalibriert;D=Hubgemessen; L=Softwareendsch.;-=Grundeinst.	30
securitystatus	(!)?	siehe Beschreibung	19
securityerror	(!)?	siehe Beschreibung	20
Fahrbefehle und Positionsverwaltung			
cal	!cal	Kalibrieren	34
rm	!rm	Tischhubmessen	34
delay	(?) !delay 1000	verzögert den Vektorstart um eine Sekunde	43
moa	!moa 10 10 10 (x 10)	Absolutposition anfahren X Y Z (nur X)	39
mor	!mor 4 4 4 (y 4)	Relativ-Positionieren X Y Z (nur Y)	39
m	!m	Start einer Verfahrbewegung (Strecke mit mor od.distance)	40
itm		Für Vektorstart durch ext. Signal	41
distance	(?) !distance	Setzen der Strecke für X Y Z (starten mit "m")	41
a	!a	Abbruch (Stop)	44
moc	!moc od. moc	alle Achsen werden zentriert (Mittelpunkt der Softwaregrenzen)	43
pos	(?) !pos 0 0 0 (z 0)	Position setzen oder lesen	42
clearpos	!clearpos	alle Positionswerte werden genullt (für endlos Drehachsen)	43
calpos	?calpos	gibt die Position (bezogen auf die Periode des Meßsystems) zurück, wo der Endschalter verlassen wurde.	36
refdir	(?) !refdir x 1	X-Achse fährt nach dem Befehl "ref" in positiver Richtung	38
ref	!ref	X-Achse kalibriert auf den Referenzschalter (nur bei der LSTEP möglich)	38

Befehl	Beispiel	Bemerkung	Kap.4 Seite
Joystick und Handrad			
speed	(?) !speed 5 5 5 (y 10)	Digitaler Joystick, alle Achsen drehen mit 5U/s od.Y mit 10	45
joydir	(?) !joydir 1 1 -1	Motordrehrichtung für Joystick einstellen	46
joy	(?) !joy 0 (1) (2) (3) (4)	Joystick Ein.-/Ausschalten mit oder ohne Positionszählung	47
	?joy	Rückmeldung "M" (Jostick manual aktiv)	47
joywindow	(?) !joywindow 10	Bereich einstellen i.d. sich die Achsen nicht bewegen (0-100)	48
joychangeaxis	(?) !joychangeaxis 1	Ändert die Zuordnung der AD-Joystickkanäle X- und Y- Achszuordnung wird getauscht	48
hw	(?) !hw 1 (0-4)	aktivieren des Handrades	50
hwvel	(?) !hwvel 1.00001.0000	Die max. Geschwindigkeit im Handradbetrieb = 1U/s	50
hwaccel	(?) !hwaccel 0.50.5	Die Beschleunigung im Handradbetrieb=0,5 m/s²	51

Bedienpult mit Trackball und Joyspeed-Tasten oder Trackball mit Funktionstasten			
joyspeed	(?) !joyspeed (1-3) 25	Parameter 1,2 oder 3 mit Geschwindigkeit 25 beschreiben	47
bpz	(?) !bpz 1 (1-4)	Bedienpult / Trackball 0=Aus 1= Ein, Joyspeedtasten 2= Ein, mit Trackball-Faktor, mit Joyspeedtasten 3= Ein, ohne Trackball-Faktor, mit Funktionstasten 4= Ein, mit Trackball-Faktor, mit Funktionstasten	51
bpztf	(?) !bpztf 10	Trackball-Faktor = 10 (Wertebereich = 0,01 bis 100)	52
bpzbl	(?) !bpzbl 0.01 0.01	Trackball-Back-Lash (Umkehrspiel einstellen; 1/10µ bis 15µ)	52

Befehl	Beispiel	Bemerkung	Kap.4 Seite
Endschalter (Hardware u. Software)			
lim	(?) !lim 0 10 0 10 0 10	Verfahrbereichsgrenzen für alle Achsen 0+10mm	29
limctr	(?) !limctr x 1	Bereichsüberwachung von Achse X ist aktiv	30
nosetlimit	(?) !nosetlimit 1 1 1 1	Für alle Achsen werden keine Verfahrbereichsgrenzen gesetzt	31
limmode	!?limmode	Überwachung der Softwaregrenzen	31
swpol	(?) !swpol 1 0 1 (z 1 0 1)	Polarität der Endschalter für alle Achsen od. nur Z zuweisen	31
swact	(?) !swact 1 0 1 (z 1 0 1)	Endschalter für alle Achsen od. nur Z Ein.-/Ausschalten	32
readsw	?readsw	lesen aller Endschalter-Zustände	32
Digitale und analoge Ein.- und Ausgänge			
digin	?digin od. ?digin 8	Lesen aller Eingänge od. lesen von Eingang 8	53
digout	!digout 5 1/?digout	Ausgang 5 wird auf 1 gesetzt / Zustand aller Ausgänge lesen	53
digfkt	!digfkt 7 0 / ?digfkt 9	-keine Beeinflussung von E-7/A-7 / -lesen der Funktion von E 9/A-9	54
edigin	nur bei LSTEP-44	(wie bei digin)	55
edigout	nur bei LSTEP-44	(wie bei digout)	55
edigfkt	nur bei LSTEP-44	(bei diesen E/A´s läßt sich nur die Polarität einstellen)	56
anain	?anain c 2	lesen des aktuellen Zustandes von Analogkanal 2	57
anaout	(?) !anaout c 1 0	setzt analogkanal 1 auf 0	57
Takt-Vor/Rück Eingänge			
tvr	(?) !tvr 1 1	aktiviert Takt-V/R für X + Y	73
tvrf	(?) !Tvrf 1	Faktor Takt Vor/Rück 1= 1Takt ist 1Motorincrement	74
Takt-Vor/Rück über Schnittstelle			
px, nx	px	1 Takt in positive Richtung in X	74
Takt-Vor/Rück Ausgänge für weitere Achsen			
tvrout	(?) !tvrout 1 1	für X + Y ist Takt-V/R Ausgang aktiv	75
tvrores	(?) !tvrores y 1000	für Y wird eine Auflösung von 1000 Impulsen/Umdr. eingestellt	75
tvropitch	(?) !tvropitch 1	bei der X-Achse wird eine 1mm Spindel eingesetzt	76
tvroa	(?) !tvroa 1	die Beschleunigung für die X-Achse ist 1m/s²	76
tvrov	(?) !tvrov z 10	die Geschwindigkeit für Z ist 10 Umdr./sec.	76
tvropos	(!) ?tvropos	alle aktuellen Positionswerte werden angezeigt	77
tvromoa	!tvromoa 10 10	X + Y werden absolut auf 10 10 gefahren	77
tvromor	!tvromor 10 10	X + Y werden relativ um 10 10 weitergefahren	77
tvrostatus	?tvrostatus	liefert den aktuellen Status: "-"=AUS "M"=Motion "@"=Steht	78

Befehl	Beispiel	Bemerkung	Kap.4 Seite
Geber-Einstellungen			
twi	(?) !twi 10 10 10	Das Zielfenster für alle Achsen=10µ (bei Dimmension=1)	65
encmask	(?) !encmask 1 0 1	Geber: X+Z aktiv; Y deaktiviert	59
enc	(?) enc	Antwort: 1 0= Geber-X = aktiv Geber-Y = deaktiviert	59
encperiod	(?) !encperiod 0.1	Teilungsperiode des X-Gebers = 0,1mm	60
encres	(?) !encres	Gibt die Anzahl der Encodersignalperioden pro Motorumdrehung an.	60
encref	(?) !encref 0	keine Referenzsignalauswertung	61
encpos	(?) !encpos 1	Bei der Positionsabfrage werden die Geberwerte angezeigt	61
encerr	(?) !encerr 0	Clear Geberfehlermeldung X; (Rückmeldung= 0 od. e)	62
ctr	(?) !ctr 2 2 2	Der Regler für alle Achsen bleibt immer an	65
ctrc	(?) !ctrc 10	Regleraufruf alle 10 ms	66
ctrs	(?) !ctrs 9 9 9	Setzt die Reglerschritte auf 9 MI/ms für alle Achsen	66
ctrf	(?) !ctrf 2 2 2	Setzt den Reglerfaktor für alle Achsen auf 2	67
ctrd	(?) !ctrd 5 5 5	Setzt die Verzögerung für alle Achsen auf 5 ms	67
ctrt	(?) !ctrt 1-10000	Reglerüberwachung (Timeout)	66
ctrfm	(?) !ctrfm 1	wenn Regeldifferenz größer Fangbereich dann neuer Vektor	68
ctrfmc	(?) !ctrfm 0	Clear Fast Move Counter / (?ctrfm = Abfrage Counter	68
MR-spezifisch			
mro	(!) ?mro	Gibt die ermittelten Offsetwerte der Steuerung zurück	69
mrp	(!) ?mrp	Gibt die Maximalwerte aller Meßsysteme zurück	69
mrt	(!) ?mrt x	Gibt die aktuellen Signalwerte von X zurück	70
mra	(!) ?mra y	Gibt den Verstärkungsfaktor von Y zurück	70
mrs	(!) ?mrsa x 0	Gibt die Signalform Sinus-X zurück	71
LSTEP-PCI - Geberposition			
hwcount	?hwcount	lese alle Geberpositionen	62
clearhwcount	!clearhwcount	alle Geberzähler auf Null setzen	62
Trigger-Ausgang			
trig	(?) !trig 1	Schaltet den Trigger ein	78
triga	(?) !triga X	Wählt die Achse aus, auf die getriggert werden soll (z.B. X)	78
trigm	(?) !trigm 1	Setzt den Trigger-Mode auf 1	79
trigs	(?) !trigs 4	Stellt die Trigger-Signal-Länge auf 4µs	80
trigd	(?) !trigd 1	Stellt die Trigger-Distance auf 1mm (bei Dim 2)	80
Trigoffsetone trigoffsettwo	(!) ?trigoffsetone ; (!) ?trigoffsettwo	-liefert die aktuelle Trigger Strecke für Trigger 1 -liefert die aktuelle Trigger Strecke für Trigger 2	81
Trigcount trigcounttwo	(!)?!trigcount; (!)?trigcounttwo	-lese Zählerstand Trigger 1 -lese Zählerstand Trigger 2	81

Befehl	Beispiel	Bemerkung	Kap.4 Seite
Snapshot-Eingang			
sns	(?) !sns 1	Snapshot "EIN"	82
snsl	(?) !snsl 1	Snapshot ist high-aktiv	82
snsm	(?) !snsm 1	Auto-Snapshot	83
snsc	?snsc	Liefert die Anzahl der ausgelösten SnapShots	83
snsp	(!) ?snsp	Liefert die gespeicherte Position	83
snsa	?snsa 11	Abfrage der Snapshot-Position 11	84
snsf	(?)!snsf 10	Dient als Eingangsfilter bei prellenden Schaltern (Wert 0-100)	82
snso	(?)!snso	Snapshot Offset	84

Erklärungen	
!	nur schreiben ("!" kann auch weggelassen werden)
(?) !	schreiben und lesen
?	nur lesen ("?" kann auch weggelassen werden)

Eingabemöglichkeiten	
Befehl Wert Wert Wert Wert	alle Achsen werden gesetzt oder gelesen
Befehl Wert Wert	nur X + Y werden gesetzt oder gelesen
Befehl Achse Wert	nur die ausgewälte Achse wird gesetzt oder gelesen

Fehlermeldungen	
0	Kein Fehler
1	Keine gültige Achsenbezeichnung
2	Keine ausführbare Funktion
3	Zu viele Zeichen im Befehls-String
4	Kein gültiger Befehl
5	Außerhalb des gültigen Zahlenbereichs
6	Falsche Anzahl der Parameter
7	Kein ! Oder ?
8	Kein TVR möglich, da Achse aktiv
9	Kein Ein- oder Ausschalten der Achsen, da TVR aktiv
10	Funktion nicht konfiguriert
11	Kein Move-Befehl möglich, da Joystick-Hand
12	Endschalter betätigt
13	Funktion kann nicht ausgeführt werden, da Encoder erkannt
14	Fehler beim Kalibrieren (Endschalter nicht korrekt freigefahren)
15	Wird beim Freifahren des Endschalters beim Kalibrieren oder Tischhubmessen der gegenüberliegende Endschalter aktiv, wird diese Funktion abgebrochen.
20	Treiberrelais defekt (Sicherheitskreis K3/K4)
21	Es dürfen nur einzelne Vektoren verfahren werden (Einrichtbetrieb)
22	Es darf kein Kalibrieren, Tischhubmessen oder Joystickbetrieb durchgeführt werden.(Tür offen oder Einrichtbetrieb)
23	SECURITY Error X-Achse
24	SECURITY Error Y-Achse
25	SECURITY Error Z-Achse

26	SECURITY Error A-Achse
27	Not-STOP
28	Fehler im Türschaltersicherheitskreis (nur bei LS44/Solero)
29	Endstufen nicht eingeschaltet (nur bei LS44)
30	GAL Sicherheitsfehler (nur bei LS44)
31	Beim Einschalten des Joy-Sticks, wenn noch ein Move aktiv ist

4.2 Informationen über die Firmware und Hardware

Mit dem Befehl „ver“ kann die Version der Firmware abgefragt werden. Im speziellen kann mit dem Befehl „det“ abgefragt werden welche Optionen in der Firmware freigeschaltet sind. Jede LStep ist durch eine interne Seriennummer eindeutig gekennzeichnet. Diese Seriennummer kann mit dem Befehl „readsn“ ausgelesen werden.

Befehl **Beispiel** **Bemerkung**

Versionsnummer auslesen	
Befehl:	?ver oder ver
Parameter:	keine
Beschreibung:	liefert die aktuelle Versionsnummer der Firmware zurück
Rückmeldung:	LS44.xx.xxx
Fehlercode:	--
Beispiel:	?ver

Interne Versionsnummer auslesen	
Befehl:	?iver oder iver
Parameter:	keine
Beschreibung:	liefert detaliertere Informationen zur Versionsnummer
Rückmeldung:	Wochentag_Kalenderwoche_Jahr-fortlaufende Nummer
Fehlercode:	--
Beispiel:	?iver Rückmeldung z. B.: T04_35-02-0004

Versionsnummer detailliert auslesen		
Befehl:		?det oder det
Parameter:		keine
Beschreibung:		Liefert die detaillierte Versionsnummer der Firmware zurück-
Rückmeldung:		Es wird ein Dezimalwert zurück geliefert, der in einen Hexadezimalwert gewandelt werden muß:
		0x0 - - - 1 ➔ 1Vss-Geber konfiguriert
		0x0 - - - 2 ➔ MR-Geber konfiguriert
		0x0 - - - 4 ➔ TTL-Geber konfiguriert
		0x0 - - 3 - ➔ Die zweite Zahl gibt die Achsenanzahl an (hier 3)
		0x0 - 1 - - ➔ Display konfiguriert
		0x0 - 2 - - ➔ Speedpoti konfiguriert
		0x0 - 4 - - ➔ Handrad konfiguriert
		0x0 - 8 - - ➔ Snapshot konfiguriert
		0x01 - - - ➔ TVRin konfiguriert
		0x02 - - - ➔ Triggerout konfiguriert
		0x08 - - - ➔ TVRout konfiguriert
		0x1 - - - - ➔ 16 digitale I/O konfiguriert
		0x2 - - - - ➔ 32 digitale I/O konfiguriert
		0x4 - - - - ➔ Trackball
		Die Kombination dieser Informationen ergibt die aktuelle Konfiguration.
Fehlercode:		--
Beispiel:		?det = 81697 ➔ $13F21_H$
Erklärung **13F21**	**1**	16 digitale I/O konfiguriert
	3	TVR und Triggerout konfiguriert
	F	Display; Speedpoti; Handrad und Snapshot konfiguriert
	2	2 Achsen
	1	1Vss Geber konfiguriert

Seriennummer lesen	
Befehl:	?readsn
Parameter:	keine
Beschreibung:	?readsn = Welche Seriennummer?
Rückmeldung:	9-Zeichen
Fehlercode:	--
Beispiel:	?readsn

4.3 Reset

Es gibt drei Möglichkeiten das Steuerungsprogramm zurückzusetzen:

- Den Hardware-Reset über den Netzschalter (bei Steuerungen ohne Anzeige).
- Den Hardware-Reset über den Reset-Taster (nur bei Steuerungen mit Anzeige).
- Den Hardware-Reset über den Dip-Schalter 1 bei der PCI-Karte
- Den Software Reset

Software - Reset	
Befehl:	Reset
Parameter:	keine
Beschreibung:	Die Steuerung wird in den Startzustand versetzt
Rückmeldung:	keine
Fehlercode:	--
Beispiel:	Reset

Poweramplifier	
Befehl:	!poweramplifier oder !pa
Parameter:	0 oder 1
Beschreibung:	Diesen Befehl gibt es nur bei LS44-Steuerungen. !poweramplifier oder !pa schaltet bei der LS44 die Endstufen ein (1) oder aus (0).
Rückmeldung:	--
Fehlercode:	--
Beispiel:	!pa 1 => Einschalten aller Endstufen der LS44.

4.4 Schnittstellenkonfiguration

Baud - Rate	
Befehl:	!baud oder ?baud
Parameter:	9600, 19200, 38400, 57600
Beschreibung:	!baud 19200 ➔ Die Übertragungsrate der Schnittstelle wird auf 19200 baud eingestellt.
	?baud ➔ liefert die aktuelle Übertragungsrate
Rückmeldung:	Aktuelle Übertragungsrate
Fehlercode:	--
Beispiel:	?baud

CTS Auswertung der RS232-Schnittstelle	
Befehl:	?cts oder !cts
Parameter:	0 oder 1
Beschreibung:	!cts 1 => aktiviert die CTS Auswertung der RS232-Schnittstelle !cts 0 => deaktiviert die CTS Auswertung der RS232-Schnittstelle ?cts => Anzeige des aktuellen Zustandes der CTS-Auswertung
Rückmeldung:	0 oder 1
Fehlercode:	--
Beispiel:	?cts (Anzeige des aktuellen Zustandes der CTS-Auswertung)

Interruptgesteuerte Kommunikation (relevant nur für LStepPCI)	
Befehl:	!intcom oder ?intcom
Parameter:	0, 1
Beschreibung:	!intcom 0 ➔ Kommunikation mit DPRAM per Polling
	!intcom 1 ➔ Kommunikation mit DPRAM Interruptgesteuert
Rückmeldung:	0 oder 1
Fehlercode:	--
Beispiel:	!intcom 1 (Umschaltung auf Kommunikation per Interrupt) ?intcom

4.5 Verwendeter Befehlssatz und Save Funktion

Die Steuerung unterstützt drei verschiedene Befehlssätze.

- Den ab der neuen Steuerungsgeneration „Juni 2000“ eingeführten und hier beschriebenen Befehlssatz.
- Den bis Juni 2000 verwendeten Register-Befehlssatz der Vorgänger-Steuerung.
- Den Multicontrol-Befehlssatz (Venus).

Mit dem folgend beschriebenen Befehl kann der gewünschte Befehlssatz gewählt werden.

Interpreter		
Befehl:	!ipreter oder ?ipreter	
Parameter:	0, 1 und 2	
Beschreibung:	!ipreter 0	➔ Register orientierter Befehlssatz
	!ipreter 1	➔ Neuer Befehlssatz
	!ipreter 2	➔ Befehlssatz ITK
	?ipreter	➔ Welcher Befehlssatz ? (kann nur 1 sein)
Zusatz:	Im Registerbefehlssatz lauten die Befehle:	
	U7ma	➔ Register
	U7mb	➔ Neuer Befehlssatz ?
	U7mc	➔ Venus Befehlssatz ?
Rückmeldung:	0, 1 oder 2	
Fehlercode:	--	
Beispiel:	!ipreter 0 (Umschaltung auf alten Befehlssatz) ?ipreter	

Der verwendete Befehlssatz wird Werksseitig eingestellt, d.h. falls aus Kompatibilitäts-gründen der ältere Register-Befehlssatz eingestellt ist kann mit folgendem Befehl auf den neueren Befehlssatz umgeschaltet werden.

Befehl: U7mb ↵

Konfiguration des Befehlssatzes	
Befehl:	!?saveipreter
Parameter:	0 (Register), 1 (Interpreter) oder 2 (ITK)
Bemerkung:	Diesen Befehl gibt es nur bei 168‘er Controllern und bedarf der „save“-Funktion, mit anschließendem RESET/NEUSTART, um wirksam zu werden.
Beschreibung:	?saveipreter => Lesen des aktuellen Zustandes !saveipreter 0 => Registersatz wird konfiguriert
Rückmeldung:	0, 1 oder 2
Fehlercode:	--
Beispiel:	!saveipreter 0 (Register-Befehlssatz) !save (Einstellung wird ins Flash gebrannt) !reset (NEUSTART)

Parameter Save Funktion	
Befehl:	Save
Parameter:	--
Bemerkung:	Diesen Befehl gibt es nur bei Steuerungen mit ST10F168 Controllern! Save bedeutet: Die aktuellen Parameter (Spindelsteigung, usw.) werden in das Flash programmiert und stehen bei einem Neustart sofort zur Verfügung.
Beschreibung:	save => Die aktuellen Parameter werden ins Flash programmiert.
Rückmeldung:	Anzeige im Display Mit ?err kann der Erfolg kontrolliert werden. D.h.: Rückmeldung = 0 => Save OK Rückmeldung ungleich 0 => Save nicht OK (siehe Controller-Manual)
Fehlercode:	--
Beispiel:	--

4.6 Status und Fehlermeldungen

AutoStatus	
Befehl:	!autostatus oder ?autostatus
Parameter:	0,1, 2, 3 oder 4
Beschreibung:	0 ➔ Es wird kein Status von der Steuerung gesendet.
	1 ➔ Es werden automatisch „Positionerreicht“ Meldungen von der Steuerung gesendet.
	2 ➔ Es werden automatisch „Positionerreicht“- und Status-Meldungen von der Steuerung gesendet.
	3 ➔ Bei „Positionerreicht“ wird nur ein Carriage Return zurückgegeben.
	4 ➔ Liefert alle Schreibbefehle mit Parametern zurück.
Rückmeldung:	
Fehlercode:	--
Beispiel:	!autostatus 1 ?autostatus

Status	
Befehl:	?status oder status
Parameter:	--
Beschreibung:	Status liefert den aktuellen Zustand der Steuerung
Rückmeldung:	OK... oder ERR und Fehlermeldung
Fehlercode:	--
Beispiel:	?status

StatusAxis		
Befehl:	?statusaxis oder statusaxis	
Parameter:	--	
Beschreibung:	Statusaxis liefert den aktuellen Zustand der einzelnen Achsen.	
Rückmeldung:	z.B.: @ - M –	
	@	➔ Achse steht und ist bereit
	M	➔ Achse ist in Bewegung (Motion)
	J	➔ Joystick-Betrieb
	C	➔ in Regelung
	S	➔ Endschalter betätigt
	A	➔ Rückmeldung nach dem Kalibrieren
	E	➔ Rückmeldung nach dem Kalibrieren wenn ein Fehler aufgetreten ist. (Endschalter nicht korrekt freigefahren)
	D	➔ Rückmeldung nach dem Tischhubmessen
	U	➔ Einrichtbetrieb (Setting **U**p)
	T	➔ Timeout
	F	➔ bei Notstopp aktiv, wenn beim Kalibrieren beide Endschalter aktiv sind, wenn beim Kalibrieren beim Herausfahren der Endlagenschalter angefahren wird
	-	➔ Achse ist nicht freigegeben
	--	
Fehlercode:	?statusaxis	
Beispiel:		

Error	
Befehl:	?err oder err
Parameter:	--
Beschreibung:	Error liefert die aktuelle Fehlernummer (siehe Beschreibung der Fehlermeldungen)
Rückmeldung:	Dezimaler Wert
Fehlercode:	--
Beispiel:	?err

Error_Nr	Beschreibung der Fehlermeldungen
0	Kein Fehler
1	Keine gültige Achsenbezeichnung
2	Keine ausführbare Funktion
3	Zu viele Zeichen im Befehls-String
4	Kein gültiger Befehl
5	Außerhalb des gültigen Zahlenbereichs
6	Falsche Anzahl der Parameter
7	Kein ! oder ?
8	Kein TVR möglich, da Achse aktiv
9	Kein Ein- oder Ausschalten der Achsen, da TVR aktiv
10	Funktion nicht konfiguriert
11	Kein Move – Befehl möglich, da Joystick – Hand
12	Endschalter betätigt
13	Funktion kann nicht ausgeführt werden, da Encoder erkannt (clear pos.)
14	Fehler beim Kalibrieren (Endschalter nicht korrekt freigefahren)
15	Wird beim Freifahren des Endschalters beim Kalibrieren oder Tischhubmessen der gegenüberliegende Endschalter aktiv, wird diese Funktion abgebrochen.
20	Treiberrelaise defekt (Sicherheitskreis K3/K4)
21	Es dürfen nur einzelne Vektoren verfahren werden (Einrichtbetrieb)
22	Es darf kein Kalibrieren, Tischhubmessen oder Joystick-Betrieb durchgeführt werden (Tür offen oder Einrichtbetrieb)
23	SECURITY Error X-Achse
24	SECURITY Error Y-Achse
25	SECURITY Error Z-Achse
26	SECURITY Error A-Achse
27	NOT-STOP
28	Fehler im Türschaltersicherheitskreis (nur bei LS44/Solero)
29	Endstufen nicht eingeschaltet (nur bei LS44)
30	GAL Sicherheitsfehler (nur bei LS44)
31	Beim Einschalten des Joy-Sticks, wenn noch ein Move aktiv ist
32	Wenn ein Move außerhalb der Softwaregrenzen liegt und Limmode=1 ist

SECURITY STATUS	
Befehl:	!?securitystatus
Parameter:	--
Bemerkung:	Diesen Befehl gibt es nur bei LS44-Steuerungen
Beschreibung:	?securitystaus => Lesen des aktuellen Zustandes der Sichheitsüberwachung !securitystatus 0 => Clear Merker für Selbsttest
Rückmeldung:	Bit 0 0000000000000000 Bit15 Bit 0-3 interne Merker Bit 4 X-Achse Stillstandsüberwachung getestet Bit 5 Y-Achse Stillstandsüberwachung getestet Bit 6 Z-Achse Stillstandsüberwachung getestet Bit 7 A-Achse Stillstandsüberwachung getestet Bit 8 X-Achse Geschwindigkeitsüberwachung getestet Bit 9 Y-Achse Geschwindigkeitsüberwachung getestet Bit 10 Z-Achse Geschwindigkeitsüberwachung getestet Bit 11 A-Achse Geschwindigkeitsüberwachung getestet Bit 12 Bit 13 Bit 14 Zustand Einrichtbetrieb (Einrichtbetrieb = 1) Bit 15 Zustand Tür (Tür „Auf“ = 1)
Fehlercode:	--
Beispiel:	--

SECURITY ERROR	
Befehl:	?securityerror
Parameter:	--
Bemerkung:	Diesen Befehl gibt es nur bei LS44-Steuerungen
Beschreibung:	?securityerror => Lesen aller Zustände und Ergebnisse der GAL-Sichheitsüberwachung
Rückmeldung:	Bit 0 0000000000000000 Bit15 Bit 0 : Achse X Stillstandsüberwachungsergebnis (OK [1] / nicht OK [0]) Bit 1 : Achse Y Stillstandsüberwachungsergebnis Bit 2 : Achse Z Stillstandsüberwachungsergebnis Bit 3 : Achse A Stillstandsüberwachungsergebnis Bit 4 : Achse X Stillstandsüberwachungstest (getestet [1] / nicht getestet [0] Bit 5 : Achse Y Stillstandsüberwachungstest Bit 6 : Achse Z Stillstandsüberwachungstest Bit 7 : Achse A Stillstandsüberwachungstest Bit 8 : Achse X Geschwindigkeitsüberwachungsergebnis (OK [1] / nicht OK [0] Bit 9 : Achse Y Geschwindigkeitsüberwachungsergebnis Bit 10 : Achse Z Geschwindigkeitsüberwachungsergebnis Bit 11 : Achse A Geschwindigkeitsüberwachungsergebnis Bit 12 : Achse X Geschwindigkeitsüberwachungstest (getestet [1] / nicht getestet [0] Bit 13 : Achse Y Geschwindigkeitsüberwachungstest Bit 14 : Achse Z Geschwindigkeitsüberwachungstest Bit 15 : Achse A Geschwindigkeitsüberwachungstest
Fehlercode:	--
Beispiel:	--

Output Function Level	
Befehl:	!?opfl
Parameter:	X, y, z, oder a 2,5-65 Umdrehungen/Sekunde und auch größer!
Bemerkung:	Beim überschreiten der eingestellten Geschwindigkeit erfolgt eine Umschaltung des Stroms, von parametriertem Strom auf maximalen Strom.
Beschreibung:	!opfl x 25 => Bei der x-Achse erfolgt eine Stromumschaltung bei 25 U/s. ?opfl => Es werden alle Geschwindigkeitsgrenzen gelesen.
Rückmeldung:	Geschwindigkeit in U/s
Fehlercode:	--
Beispiel:	?opfl y (Lese die Geschwindigkeitsgrenze der y-Achse)

Switch Level for Velocity	
Befehl:	!?vlevel
Parameter:	1-7 und 0 – max. Geschwindigkeit
Bemerkung:	Mit diesem Befehl können Geschwindigkeitsbereiche ausgeklammert werden, in denen das System zu Resonanzen neigt. Es gibt 3 Geschwindigkeitsbereiche und eine Grenze die mit diesem Befehl eingestellt werden können: Vlevel 1 = Untere Grenze des ersten/unteren Bereiches Vlevel 2 = Obere Grenze des ersten/unteren Bereiches Vlevel 3 = Untere Grenze des zweiten/mittleren Bereiches Vlevel 4 = Obere Grenze des zweiten/mittleren Bereiches Vlevel 5 = Untere Grenze des dritten/oberen Bereiches Vlevel 6 = Obere Grenze des dritten/oberen Bereiches Vlevel 7 = Bis zu dieser Geschwindigkeitsgrenze wird die Korrekturtabelle genutzt. Gilt für alle Achsen!
Beschreibung:	!vlevel 1 0.8 = Untere Grenze des ersten/unteren Bereiches !vlevel 2 1.2 = Obere Grenze des ersten/unteren Bereiches. ?vlevel 3 = Lesen der Geschwindigkeitsgrenze des zweiten/unteren Bereiches
Rückmeldung:	Eingestellte Geschwindigkeit
Fehlercode:	--
Beispiel:	!vlevel 7 10 (Die Korrekturtabelle wirkt bis zu einer Geschwindigkeit von 10 Umdrehungen/s)

Motor – Table Patch	
Befehl:	!?mtpatch
Parameter:	0 der 1
Bemerkung:	Mit diesem Befehl wird die Korrekturtabelle aktiviert. Die Korrekturtabelle wurde für einen Sondermotor durch Meßung ermittelt. Korrekturtabellen können auf Kundenwunsch ermittelt werden.
Beschreibung:	!mtpatch 1 = Aktivierung der Korrekturtabelle ?mtpatch = Lesen des aktuellen Zustands
Rückmeldung:	0 oder 1
Fehlercode:	--
Beispiel:	!mtpatch 0 (Die Korrekturtabelle wird nicht benutzt.)

Joystick Filter	
Befehl:	!?joyfilter
Parameter:	0 der 1
Bemerkung:	Mit diesem Befehl wird die Filterung und Hysterese im Joystick-Betrieb aktiviert
Beschreibung:	!joyfilter 1 = Aktivierung der Filterung ?joyfilter = Lesen des aktuellen Zustands
Rückmeldung:	0 oder 1
Fehlercode:	--
Beispiel:	!joyfilter 0 (Die Filterung und Hysterese wird nicht benutzt.)

4.7 Einstellungen

Mit den folgend Beschriebenen Befehlen kann die Steuerung an die eingesetzte Mechanik und die gewünschten Anforderungen angepasst werden.

Dimension	
Befehl:	!dim oder ?dim
Parameter:	X, y, z und a 0, 1, 2, 3 oder 4 Die Einheiten von Längenangaben bei Ein- und Ausgabe sind:
	0 ➔ Microsteps
	1 ➔ µm
	2 ➔ mm
	3 ➔ 360°
	4 ➔ Anzahl der Umdrehungen
Beschreibung:	
	!dim 4 1 ➔ Die Dimensionen für x- und y-Achse sind „Anzahl der Umdrehungen„ und „µm„.
	!dim z 2 ➔ Die Dimensionen für die z-Achse ist „mm„.
	?dim ➔ Es werden alle Dimensionen angezeigt.
	?dim a ➔ Es wird die Dimension der a-Achse angezeigt.
Rückmeldung:	Aktuelle Einstellung
Fehlercode:	--
Beispiel:	!dim 1 1 1 1 (Alle Werte in µm) ?dim

Hinweis:	Für die Dimension 3 (Grad) und 4 (Umdrehungen) sollte die Spindelsteigung auf 1 mm eingestellt werden.

Spindelsteigung		
Befehl:	!pitch oder ?pitch	
Parameter:	X, y, z und a 0.0001 – 68	
Beschreibung:	!pitch 4.0 1.0	➔ Spindelsteigungen x = 4mm und y = 1mm werden programmiert.
	!pitch z 1.0	➔ Spindelsteigung z = 1mm wird programmiert.
	?pitch	➔ Es werden alle Spindelsteigungen angezeigt.
	?pitch a	➔ Es wird die Spindelsteigung der a-Achse angezeigt.
Rückmeldung:	Aktuelle Spindelsteigung	
Fehlercode:	--	
Beispiel:	!pitch 10 (Spindelsteigung x = 10mm) ?pitch	

Getriebe		
Befehl:	!gear oder ?gear	
Parameter:	X, y, z und a 0.01 – 0.99 und 1-1000	
Beschreibung:	!gear 4.0 1.0	➔ Getriebe-Übersetzungen ¼ bei x und 1/1 bei y werden programmiert.
	!gear z 10.0	➔ Getriebe-Übersetzungen 1/10 bei z wird programmiert.
	?gear	➔ Es werden alle Getriebe-Übersetzungen angezeigt.
	?gear a	➔ Es wird die Getriebe-Übersetzungen der a-Achse angezeigt.
Rückmeldung:	Aktuelle Getriebe-Übersetzungen	
Fehlercode:	--	
Beispiel:	!gear 10 (Getriebe-Übersetzung 1/10 bei x) ?gear	

Beschleunigung		
Befehl:	!accel oder ?accel	
Parameter:	X, y, z und a 0.01 – 20.00 [m/s²]	
Beschreibung:	!accel 1.00 1.50	➔ Bei Achse x und y werden die Beschleunigungen (x=1.00, y=1.50 [m/s²]) eingestellt, die anderen Achsen bleiben unverändert.
	!accel x 1	➔ Die Beschleunigung für Achse x wird auf 1.00 [m/s²] eingestellt.
	?accel	➔ Alle eingestellten Beschleunigungen werden angezeigt.
	?accel z	➔ Die eingestellte Beschleunigung der z-Achse wird angezeigt.
Rückmeldung:	Eingestellte Beschleunigung	
Fehlercode:	--	
Beispiel:	!accel 1.00 (Setze Beschleunigungen bei x-Achse auf 1 m/s²) ?accel	

Geschwindigkeit		
Befehl:	!vel oder ?vel	
Parameter:	X, y, z und a 0 – maximale Geschwindigkeit	
Beschreibung:	!vel 1.0 15	➔ Bei Achse x und y werden die Geschwindigkeitswerte (x=1.0, y=15 [U/s]) beschrieben, die anderen Achsen bleiben unverändert.
	!vel z 0.1	➔ Bei der z-Achse wird die Geschwindigkeit auf 0.1 [U/s] eingestellt.
	?vel	➔ Alle eingestellten Geschwindigkeiten werden angezeigt.
	?vel x	➔ Anzeigen der eingestellten Geschwindigkeit von Achse x.
Rückmeldung:	Eingestellte Geschwindigkeit	
Fehlercode:	--	
Beispiel:	!vel 10 (Die x-Achse wird mit maximal 10 U/s betrieben) ?vel	

Die Drehzahl der Motoren sind in Stufen (St) von 0,01 U/sec. bis 40 U/sec, bzw. bei der ECO-STEP bis 15 U/sec. einstellbar. Die oberen Drehzahlbereiche lassen sich nur bei optimaler Abstimmung der Motoren und Mechanik an die LSTEP erreichen.

Wert	Drehzahl [U/sec]	Wert	Drehzahl [U/sec]	Wert	Drehzahl [U/sec]
0	0.01	2.0	2.0	12.0	12
0.1	0.1	3.0	3.0	13.0	13
0.2	0.2	9.0	9.0	15.0	15
0.9	0.9	10.0	10	20.0	20
1.0	1.0	11.0	11	40.0	40

Geschwindigkeitsuntersetzung	
Befehl:	!velfac ?velfac
Parameter:	X, y, z oder a 0.01 bis 1.00
Beschreibung:	!velfac x 0.1 => reduziert die Geschwindigkeit der X-Achse auf 1/10 der eingestellten Geschwindigkeit. ?velfac => liefert die Einstellungen aller Achsen
Rückmeldung:	Es wird ein Dezimalwert zurück geliefert (0.01 bis 1.00)
Fehlercode:	--
Beispiel:	?velfac z (liefert die Einstellung der Z-Achse)

MaxCurrent (max. möglicher Motorstrom)	
Befehl:	?maxcur
Parameter:	X, y, z oder a
Beschreibung:	?maxcur y => liefert den maximal möglichen Motorstrom der Y – Achse ?maxcur => liefert den maximal möglichen Motorstrom aller Achsen
Rückmeldung:	Motorstrom in Ampere
Fehlercode:	--
Beispiel:	?maxcur

Ausgangsstrom	
Befehl:	!cur oder ?cur
Parameter:	X, y, z und a 0 – maximaler Strom
Beschreibung:	!cur 1.0 2 ➔ Bei den Achsen x und y werden die Ausgangsströme auf x = 1A und y = 2A eingestellt, die anderen Achsen bleiben unverändert:
	!cur z 0.1 ➔ Bei der z-Achse wird der Ausgangsstrom auf 0.1A eingestellt.
	?cur ➔ Alle eingestellten Ausgangsströme werden angezeigt.
	?cur x ➔ Anzeigen des eingestellten Ausgangsstroms von Achse x.
Rückmeldung:	Eingestellter Ausgangsstrom
Fehlercode:	--
Beispiel:	!cur 1.0 (Die x-Achse wird mit maximal 1A betrieben) ?cur

Stromabsenkung	
Befehl:	!reduction oder ?reduction
Parameter:	X, y, z und a 0 – 1.0
Beschreibung:	Im Ruhezustand wird der Motornennrstrom auf das parametrierte Verhältnis reduziert.
	!reduction 0.1 .7 ➔ x-Achse = 0.1*Nennstrom und y-Achse = 0.7*Nennstrom
	!reduction z 0.5 ➔ z-Achse = 0.5*Nennstrom
	?reduction ➔ Anzeige der eingestellten Stromabsenkungen aller Achsen
	?reduction x ➔ Anzeige der eingestellten Stromabsenkung der Achse x.
Rückmeldung:	Eingestellte Stromabsenkung
Fehlercode:	--
Beispiel:	!reduction 0.3 0.5 (x- und y-Achse werden reduziert) ?reduction

Verzögerung Stromabsenkung (Delay Reduction)	
Befehl:	!curdelay oder ?curdelay
Parameter:	X, y, z und a 0 – 10000 (ms)
Beschreibung:	Nach dem Verfahren eines Vektors bleibt der Motornennstrom für die in curdelay eingestellte Zeit erhalten. Danach wird er auf den in der Stromabsenkung spezifizierten Wert abgesenkt. !curdelay 100 300 = x-Achse = 100 ms Verzögerung und y-Achse = 300 ms Verzögerung !curdelay z 450 = z-Achse = 450 ms Verzögerung ?curdelay = Anzeige der eingestellten Stromabsenkungsverzögerungen aller Achsen ?curdelay x = Anzeige der eingestellten Stromabsenkungsverzögerung von Achse x
Rückmeldung:	Eingestellte Verzögerung der Stromabsenkung
Fehlercode:	--
Beispiel:	!curdelay 100 300 (x- und y-Achse werden verzögert) ?curdelay

Achsenfreigabe		
Befehl:	!axis oder ?axis	
Parameter:	x, y, z und a 0 und 1	
Beschreibung:	!axis 1 0 1 0	➔ Achse x und z sind freigegeben, Achse y und a sind nicht freigegeben.
	!axis y 1	➔ Achse y freigeben
	?axis	➔ Zustand aller Achsen darstellen
	?axis a	➔ Zustand Achse a darstellen
Rückmeldung:	Aktueller Betriebszustand	
Fehlercode:	--	
Beispiel:	!axis 1 1 1 1 (alle Achsen freigeben) ?axis x (lese Zustand der x-Achse)	

Axis direction	
Befehl:	!?axisdir
Parameter:	X, y, z und a 0 oder 1
Bemerkung:	Mit axisdir können die Motor - Drehrichtungen gedreht werden, die dazugehörigen Endschalter werden mit gedreht.
Beschreibung:	!axis 0 1 0 1 => Bei den Achsen y und a werden die Drehrichtungen gedreht. ?axisdir x = Anzeige, ob bei der x-Achse die Drehrichtung aktiviert ist.
Rückmeldung:	0 = Kein Drehrichtungswechsel 1 = Drehrichtungswechsel
Fehlercode:	--
Beispiel:	!axisdir 0 0 0 0 (Aufheben aller Drehrichtungswechsel)

Limit		
Befehl:	!lim oder ?lim	
Parameter:	x, y, z oder a +- maximale Verfahrbereich	
Bemerkung:	Die Werte müssen paarweise vorgegeben werden. Die Ein- und Ausgabewerte sind abhängig von der Dimension.	
Beschreibung:	!lim -1000 1000 -2000 2000	➔ Achse x und y werden Verfahrbereichsgrenzen zugewiesen.
	!lim z –500 1700	➔ Achse z Verfahrbereichsgrenzen zuweisen.
	?lim	➔ Verfahrbereichsgrenzen aller Achsen lesen
	?lim a	➔ Verfahrbereichsgrenze Achse a darstellen
Rückmeldung:	Aktuelle Verfahrbereiche	
Fehlercode:	--	
Beispiel:	!lim 10 (nur untere Grenze x-Achse programmieren) ?lim	

Bereichsüberwachung	
Befehl:	!limctr oder ?limctr
Parameter:	x, y, z oder a 0 oder 1
Beschreibung:	!limctr 1 1 1 ➔ Bereichsüberwachung von x-, y- und z-Achse aktiv.
	!limctr z 1 ➔ Bereichsüberwachung von Achse z aktiv.
	?limctr a ➔ Bereichsüberwachung Achse a aktiv ?
	?limctr Anzeige des Zustands der einzelnen Bereichsüberwachungen.
Rückmeldung:	0 = Bereichsüberwachung nicht aktiv 1 = Bereichsüberwachung aktiv
Fehlercode:	--
Beispiel:	! limctr y 0 (Bereichsüberwachung der Achse y deaktivieren ? limctr

Statuslimit	
Befehl:	?statuslimit oder statuslimit
Parameter:	--
Beschreibung:	Statuslimit liefert den aktuellen Zustand der Software-Grenzen jeder einzelnen Achse
Rückmeldung:	A = Achse wurde kalibriert
	D = Tischhub wurde gemessen
	L = Software – Limit wurde gesetzt
	- = Software – Grenze wurde nicht verändert
	Die Reihenfolge der Rückmeldung ist zum Beispiel: AA-A--DD-LL-L--L
	X,y und a = Kalibriert
	Z und a = Tischhub gemessen Y und z = min. Softwarelimit gesetzt
	X und a = max. Softwarelimit gesetzt
Fehlercode:	--
Beispiel:	?statuslimit

NoSetLimit	
Befehl:	!?nosetlimit
Parameter:	x, y, z oder a 0 oder 1
Bemerkung:	Beim Kalibrieren und Tischhubmessen werden normalerweise die internen Software - Limits gesetzt, daß kann hiermit verhindert werden.
Beschreibung:	nosetlimit 1 1 1 => Bei den Achsen x, y und z werden keine Verfahrbereichsgrenzen gesetzt. !nosetlimit y 1 => Bei der Achse y wird keine Verfahrbereichsgrenze gesetzt. ?nosetlimit = Einstellung aller Achsen lesen ?nosetlimit a = Einstellung der Achse a lesen
Rückmeldung:	0 = Software – Limits werden gesetzt (calib/rm)
Fehlercode:	--
Beispiel:	?nosetlimit

Überwachung der Softwaregrenzen	
Befehl:	!?limmode
Parameter:	0+1
Beschreibung:	0 = Softwarelimits werden überwacht so wie bei den älteren Versionen 1 = Moves werden nicht ausgeführt wenn sie außerhalb des Verfahrbereichs liegen, es kommt ERR 32 zurück.
Rückmeldung:	Eingestellter Modus

Endschalterpolarität		
Befehl:	!swpol oder ?swpol	
Parameter:	x, y, z oder a **0**  oder **1** 	
Beschreibung:	!swpol 1 0 1	➔ Polarität der Endschalter aller Achsen zuweisen. (Reihenfolge: E0 REF EE).
	!swpol z 1 0 1	➔ Polarität der Endschalter von Achse z zuweisen. (Reihenfolge: E0 REF EE)
	?swpol a	➔ Polarität der Endschalter von Achse a darstellen.
Rückmeldung:	Polarität der Endschalter	
Fehlercode:	--	
Beispiel:	!swpol y 1 1 1 (Alle Schalter der Achse y reagieren auf die positive Flanke)	

	?swpol x

Endschalter Ein/Aus	
Befehl:	!swact oder ?swact
Parameter:	x, y, z oder a 0 oder 1
Beschreibung:	!swact 1 0 1 ➔ Endschalter aller Achsen : E0=Ein REF=Aus EE=Ein
	!swact z 1 0 1 ➔ Endschalter von Achse z: E0=Ein REF=Aus EE=Ein
	?swact a ➔ Zustand der Endschalter von Achse a darstellen.
Rückmeldung:	Zustand der Endschalter
Fehlercode:	--
Beispiel:	!swact y 1 1 1 (Alle Schalter der Achse y aktiv) ?swact x

Endschalter einlesen												
Befehl:	?readsw											
Parameter:												
Beschreibung:	?readsw ➔ Lesen aller Endschalterzustände.											
Rückmeldung:	Zustand der Endschalter.											
	Achse:	**x**	**y**	**z**	**a**	**x**	**y**	**z**	**a**	**x**	**y**	**z**	**a**
	Schalter:	**E0**	**E0**	**E0**	**E0**	**Ref**	**Ref**	**Ref**	**Ref**	**EE**	**EE**	**EE**	**EE**
	E0 = Null-Endschalter				Ref = Referenz-Endschalter				EE = End-Endschalter			
Fehlercode:	--											
Beispiel:	?readsw (Lesen aller Endschalter)											

Stopeingang Polarität einstellen	
Befehl:	!stoppol oder ?stoppol
Parameter:	0 = lowaktiv 1 = highaktiv
Beschreibung:	Da der Stopeingang einen Pull Up nach 5V hat, muß man bei einem Schließer lowaktiv und bei einem Öffner highaktiv einstellen.
Rückmeldung:	--
Fehlercode:	--
Beispiel:	!stoppol 1 (der Stopeingang ist highaktiv)

Bremsbeschleunigung für Not-Stop	
Befehl:	!stopaccel oder ?stopaccel
Parameter:	0,01 bis 20 m/s²
Beschreibung:	Wenn "stopaccel" nicht beschrieben wird, wird bei aktiv werden des Stopeingangs, mit der Beschleunigung die mit "accel" eingestellt wurde angehalten. Wird "stopaccel" beschrieben, gilt diese Beschleunigung, außer wenn der Wert in "accel" größer ist. Der Wert wird nicht mit Save gespeichert.Die Position geht nicht verloren (wenn die Beschleunigung richtig gewählt wurde). Nach der Freigabe des Stopeinganges muß nicht mehr Kalibriert werden.
Rückmeldung:	--
Fehlercode:	--
Beispiel:	!stopaccel 2 (es wird mit 2m/s² angehalten)
Achtung! **Bemerkung:**	stopaccel gilt nur für den Vektorbetrieb, **nicht für:** Joystick, Kalibrieren und Hubmessen

4.8 Mechanischen Arbeitsbereich ermitteln

Nach dem initialisieren der Steuerung sollten die Befehle Kalibrieren „cal“ und Hub-Messen „rm“ausgeführt werden. Dadurch wird der maximale Mechanische Arbeitsbereich ermittelt. Hierdurch ist gewärleistet, dass die Achsen nicht mehr in die Endschalter verfahren werden können.
Das messen des Arbeitshubes ist nur möglich, wenn alle Achen einen Null- sowie einen Endendschalter besitzen.
Damit beim Anfahren der Null- oder Endposition durch ein Überschwingen der Mechanik die Endschalter nicht ansprechen, kann mit den Befehlen „caliboffset“ und „rmoffset“der Arbeitsbereich eingeengt werden.

Kalibrieren	
Befehl:	!cal oder cal
Parameter:	X, y, z oder a
Beschreibung:	Cal ➔ Bewegt alle freigegebenen Achsen in Richtung kleinerer Positionswerte. Die Verfahrbewegung wird unterbrochen sobald die Endschalter angefahren wurden und dann langsam in entgegengesetzter Richtung gefahren bis der Schalter nicht mehr aktiv ist. Der Positionswert wird auf 0 gesetzt. Die Position wird als Softwaregrenze, wie unter dem Befehl „Limit“ beschrieben, übernommen.
	Cal y ➔ Wie oben jedoch nur y-Achse.
Rückmeldung:	Für jede kalibrierte Achse ein ‚A‘oder ‘E’ bei einem Fehler
Fehlercode:	--
Beispiel:	!cal

Tischhub messen	
Befehl:	!rm oder rm
Parameter:	X, y, z oder a
Beschreibung:	Rm ➔ Bewegt alle freigegebenen Achsen in Richtung größerer Positionswerte. Die Verfahrbewegung wird unterbrochen sobald die Endschalter angefahren wurden und dann langsam in entgegengesetzter Richtung gefahren bis der Schalter nicht mehr aktiv ist. Der Positionswert wird gespeichert und als Softwaregrenze, wie unter dem Befehl „Limit“ beschrieben, übernommen.
	Rm z ➔ Wie oben jedoch nur die z-Achse
Rückmeldung:	Für jede Achse ein ‚D‘
Fehlercode:	--
Beispiel:	!rm

RM Offset	
Befehl:	!rmoffset oder ?rmoffset
Parameter:	X, y, z oder a 0 – 32*50000 (32*Spindelsteigung)
Beschreibung:	!rmoffset 1 1 1 ➔ Die Achsen X, Y und Z werden beim Tischhub messen jeweils 1mm (bei Dim 2 2 2) vom Endendschalter in Richtung Tischmitte verfahren und dann die Softwaregrenze gesetzt.
	?rmoffset y ➔ Aktuellen Offset der Y-Achse lesen.
Rückmeldung:	Strecke
Fehlercode:	--
Beispiel:	?rmoffset

Kalibrier-Offset	
Befehl:	!caliboffset oder ?caliboffset
Parameter:	X, y, z oder a 0 – 32*50000 (32*Spindelsteigung)
Beschreibung:	!caliboffset 1 1 1 ➔ Die Achsen X, Y und Z werden beim Kalibrieren jeweils 1mm (bei Dim 2 2 2) vom Nullendschalter in Richtung Tischmitte verfahren und dann die Position Null gesetzt (Softwaregrenze).
	?caliboffset y ➔ Aktuellen Offset der Y-Achse lesen
Rückmeldung:	Strecke
Fehlercode:	--
Beispiel:	?caliboffset

Calibration Direction	
Befehl:	!?caldir
Parameter:	X, y, z oder a 0 oder 1
Bemerkung:	Beim kalibrieren in positiver Richtung wird das positive Software Limit gesetzt.
Beschreibung:	!caldir 0 0 1 => Die Achsen X, Y werden in negativer Richtung und die Z-Achse in positiver Richtung kalibriert. ?caldir => Lese aktuelle Richtung für das kalibrieren.
Rückmeldung	0 = negative Richtung 1 = positive Richtung
Fehlercode:	--
Beispiel:	!caldir y 1 (Die Y-Achse wird in positiver Richtung kalibriert)
Achtung!	Dieser Befehl geht nur bei Steuerungen ohne Meßsystem

Calibration Position	
Befehl:	!?calpos (nur in Verbindung mit einem Meßsystem)
Parameter:	X, y, z oder a Positionswert
Bemerkung:	Beim Kalibrieren wird für jede Achse die Position der Meßsystem-Periode gespeichert, an der der Endschalter verlassen wurde.
Beschreibung:	!calpos 0 0 0 => Setze die Positionen für X-, Y- und Z-Achse auf 0. ?calpos => Lese aktuelle Position
Rückmeldung	Im Bereich der Geber-Signalperiode
Fehlercode:	--
Beispiel:	?calpos y (Die Position der Y-Achse)

Calibration Backspeed	
Befehl:	!?calbspeed
Parameter:	Wertebereich 5 bis 100
Bemerkung:	Die Geschwindigkeit entspricht dem angegebenen Wert *0.01 U/s.
Beschreibung:	calbspeed setzt bzw. liest die Umdrehungsgeschwindigkeit, mit der die Achsen beim Kalibrieren nach dem Anfahren der Endschalter wieder herausgefahren werden. Der eingegebene Wert muss mit 0.01 U/s multipliziert werden.
Rückmeldung	--
Fehlercode:	--
Beispiel:	!calbspeed 10 => Die Endschalter werden bei der Kalibrierung nach dem Anfahren mit 0.1 U/s verlassen. ?calbspeed => Lese aktuelle Einstellung (ausgegebener Wert *0.01 U/s).

Calibration Refspeed	
Befehl:	!?calrefspeed
Parameter:	Wertebereich 0 - 100
Bemerkung:	Die Grundeinstellung = 32 Der Wert wird nicht mit Save gespeichert.
Beschreibung:	Diese Einstellung verändert die Geschwindigkeit mit der die Referenzmarke gesucht wird.
Rückmeldung	--
Fehlercode:	--
Beispiel:	!calrefspeed 5

Direction for Reference	
Befehl:	!?refdir (gilt nur für LSTEP)
Parameter:	X, y, z oder a 0 oder 1
Bemerkung:	Im Grundzustand wird bei nicht betätigtem Schalter in negativer Richtung referenziert, dies kann durch „!refdir“ verändert werden.
Beschreibung:	!refdir 0 0 1 => Die Achsen X, Y werden in negativer Richtung und die Z-Achse in positiver Richtung referenziert. ?refdir => Lese aktuelle Richtung für das referenzieren
Rückmeldung	0 = negative Richtung 1 = positive Richtung
Fehlercode:	--
Beispiel:	!refdir Y (Die Y-Achse wird in positiver Richtung referenziert)

Reference	
Befehl:	!ref oder ref (gilt nur für LSTEP)
Parameter:	X, y, z oder a
Bemerkung:	Im Grundzustand wird bei nicht betätigtem Schalter in negativer Richtung referenziert, dies kann durch „!refdir“ verändert werden.
Beschreibung:	ref = Bewegt alle freigegebenen Achsen in die durch refdir angegebene Richtung. Die Verfahrbewegung wird unterbrochen sobald die Referenzschalter angefahren wurden. Der Positionswert wird nicht gesetzt. ref y = Wie oben jedoch nur y-Achse.
Rückmeldung	Für jede referenzierte Achse ein ‚R‘
Fehlercode:	--
Beispiel:	!ref

4.9 Verfahrbefehle und deren Kontrollfunktionen

Bei allen Positionierungsbefehlen wird eine Linearinterpolation durchgeführt, d.h. alle Achsen erreichen die vorgegebene Position zum gleichen Zeitpunkt. Die Achse, wo der Motor die meisten Umdrehungen zurücklegen muß, gilt als Führungsachse und fährt somit mit der eingestellten Geschwindigkeit und Beschleunigung. Müssen alle Motoren die gleiche Strecke zurücklegen, ist die x-Achse die
Führungsachse. Hierbei kann die eingestellte Geschwindigkeit und Beschleunigung von Achsen überschritten werden.
Hat man Achsen mit total unterschiedlichen dynamischen Verhalten, sollte man diese einzeln starten. Auch ein assynchrones Verfahren ist möglich. Hierbei ist zu beachten, dass bei der Einstellung Autostatus 1 die Rückmeldung erst kommt, wenn alle Achsen stehen. Möchte man während sich eine Achse bewegt eine andere Achse mehrmals starten, setzt man den Autostatus = 0 und pollt mit ?statusaxis.

Position absolut	
Befehl:	!moa oder moa
Parameter:	X, y, z oder a +- Verfahrbereich
Bemerkung:	Die Eingabe ist abhängig von der Dimension.
Beschreibung:	Moa 10 0 20 ➔ Die Achsen x, y und z werden auf die eingegebenen Positionswerte positioniert.
	moa y 333 ➔ Wie oben jedoch nur y-Achse.
Rückmeldung:	Für jede positionierte Achse ein ‚@'
Fehlercode:	--
Beispiel:	Moa x 10 (Die x-Achse wird auf die eingegebene Position positioniert)

Position relativ	
Befehl:	!mor oder mor
Parameter:	X, y, z oder a +- Verfahrbereich
Bemerkung:	Die Eingabe ist abhängig von der Dimension.
Beschreibung:	Mor 100 0 39 ➔ Die Achsen x und z werden um die eingegebenen Strecken verfahren.
	Mor a 298 ➔ Die a-Achse wird um die eingegebene Strecke verfahren.
Rückmeldung:	Für jede verfahrene Achse ein ‚@'
Fehlercode:	--
Beispiel:	!mor 0 0 0 100 (Nur die a-Achse wird verfahren)

X Y Compensation	
Befehl:	!?xycomp
Parameter:	0 bis 6
Bemerkung:	0 = Keine Kompensation 1 = „X = X+Y“ 2 = „Y = X+Y“ 3 = „X = X-Y und Y = X+Y“ 4 = „X = X+Y und Y = X-Y“ 5 = „X = X-Y“ 6 = „Y = X-Y“
Beschreibung:	xycomp 1 => Die Achsen X, Y werden nach obiger Formel manipuliert. ?xycomp => Lese aktuellen Zustand
Rückmeldung:	Art der Kompensation
Fehlercode:	--
Beispiel:	?xycomp (Lese aktuellen Zustand der Kompensation)

Position relativ (Kurzbefehl)	
Befehl:	!m oder m
Parameter:	
Bemerkung:	Dieser Befehl wird verwendet, wenn in sehr kurzer Abfolge immer wieder die gleiche Strecke Verfahren werden soll. Die zu verfahrende Strecke muß vorher mit !distance oder mor befehlen gesetzt werden. Die Position wird nicht aktualisiert, erst wieder beim nächsten Move-Befehl.
Beschreibung:	m ➔ Start einer Verfahrbewegung aller freigegebenen Achsen.
Rückmeldung:	Je nach Einstellung von autostatus.
Fehlercode:	--
Beispiel:	!mor 0 0 0 100 (Nur die a-Achse wird verfahren) m (A-Achse wird wieder um 100 verfahren)

Externe Vectorstart	
Befehl:	!?itm
Parameter:	0-4
Bemerkung:	Verfahren wird der Wert, der in « distance » steht. Bei „autostaus = 1“, kommt nach dem Move das @@@, und itm wird auf 0 gesetzt.Bei Autostatus 0 kommt keine Rückmeldung und es können beliebig viele Vektoren gefahren werden.
Beschreibung:	0 = Funktion nicht aktiv 1 = absolut Positionieren bei positiver Flanke 2 = absolut Positionieren bei negativer Flanke 3 = relativ Positionieren bei positiver Flanke 4 = relativ Positionieren bei negativer Flanke
Rückmeldung:	Eingestellter Modus

Strecke		
Befehl:	!distance oder ?distance	
Parameter:	X,y,z und a Min-/max-Verfahrbereich	
Bemerkung:	Ein- und Ausgabe ist abhängig von der Dimension.	
Beschreibung:	!distance 1 2 3	➔ Die Strecken für die Achsen x, y und z werden gesetzt.
	!distance y 20	➔ Die Strecke der y-Achse wird gesetzt.
	?distance	➔ Abfrage der aktuellen Strecken aller Achsen.
	?distance z	➔ Abfrage der aktuellen Strecke von Achse z.
Rückmeldung:	Strecken	
Fehlercode:	--	
Beispiel:	!distance 10 20 (Setzen der Strecken von x- und y-Achse) ?distance (Abfrage der Strecken aller Achsen)	

SpeedPoti	
Befehl:	!pot oder ?pot
Parameter:	0 oder 1
Bemerkung:	
Beschreibung:	0 ➔ Es wird die vorgegebene Geschwindigkeit (vel) als Verfahrgeschwindigkeit genutzt.
	1 ➔ Es wird die vorgegebene Geschwindigkeit (vel), in Abhängigkeit von der Stellung des Potentiometers, prozentual genutzt.
Rückmeldung:	--
Fehlercode:	--
Beispiel:	!pot 1 ?pot

Position		
Befehl:	!pos oder ?pos	
Parameter:	X,y,z und a Min-/max-Verfahrbereich	
Bemerkung:	Ein- und Ausgabe ist abhängig von der Dimension.	
Beschreibung:	!pos 1000 2000 3000	➔ Die Positionswerte für die Achsen x, y und z werden gesetzt.
	!pos y 2000	➔ Die Position der y-Achse wird gesetzt.
	?pos	➔ Abfrage der aktuellen Position aller Achsen.
	?pos z	➔ Abfrage der aktuellen Position von Achse z.
Rückmeldung:	Positionswerte	
Fehlercode:	--	
Beispiel:	!pos100 200 (Setzen der Positionen von x- und y-Achse) ?pos (Abfrage der Positionen aller Achsen)	

Clear Position	
Befehl:	Clearpos
Parameter:	X,y,z und a
Bemerkung:	Dieser Befehl setzt die Position auf Ø, auch den internen Zähler (ist nicht die gleiche Funktion wie Position setzen mit !pos x Ø). Gebraucht wird diese Funktion für Endlosachsen, da die Steuerung nur ± 1000 Motorumdrehungen vom Wertebereich verarbeiten kann. Bei erkannten Gebern wird die Funktion. für die jeweilige Achse nicht ausgeführt
Beschreibung:	clearpos => Alle Positionswerte werden genullt.
	clearpos y => Position der y-Achse wird genullt.
Rückmeldung:	Keine
Fehlercode:	--
Beispiel:	clearpos x (Position der x-Achse wird genullt)

Positionierung Zentral	
Befehl:	!moc oder moc
Parameter:	X, y, z und a
Bemerkung:	Alle Achsen werden zentriert. Es ist Sinnvoll vorher zu kalibrieren und Tischhub zu messen!
Beschreibung:	Moc a => Die A-Achse wird zentriert.
Rückmeldung:	Für jede positionierte Achse ein ‚@'
Fehlercode:	--
Beispiel:	Moc (Es werden alle Achsen zentriert)

Verzögerung	
Befehl:	?delay oder !delay
Parameter:	0 – 10000 (ms)
Beschreibung:	Durch den Befehl Delay kann eine Verzögerung des Vektorstarts erzeugt werden.
Rückmeldung:	Dezimaler Wert
Fehlercode:	--
Beispiel:	!delay 1000 (1s Verzögerung) ?delay

Abbruch	
Befehl:	!a oder a
Parameter:	Keine
Beschreibung:	Es werden alle Verfahrbewegungen abgebrochen.
Rückmeldung:	Für jede Achse ein @
Fehlercode:	--
Beispiel:	!a

4.10 Joystick- Handrad- und Trackball-Befehle

Hinweis:	Ist der Joystickschalter an der Steuerung auf „manual“ gestellt, so können alle Achsen mit Hilfe des Joysticks bis in die Endlagen verfahren werden. Hierbei wird die Position mitgezählt. Befehle zur Einstellung der Steuerung sind in diesem Betriebszustand möglich, „Move“ - Befehle jedoch nicht. Bei der Abfrage mit dem Befehl „Statusaxis“ liefert die Steuerung „J J J“. Bei der Abfrage mit dem Befehl „?joy“ liefert die Steuerung ein „M“.

Digitaler Joystick (Geschwindigkeit)		
Befehl:	!speed oder ?speed	
Parameter:	x, y, z oder a +- maximale Geschwindigkeit (vel)	
Beschreibung:	Mit diesem Befehl können einzelne Achsen mit einer konstanten Geschwindigkeit verfahren werden.	
	!speed 0	➔ Alle Achsen Geschwindigkeit 0 und Joystick-Betrieb „AUS„
	!speed 10	➔ Alle Achsen Geschwindigkeit 10.0 [U/s] und Joystick-Betrieb „EIN„.
	!speed 10 10 0 10	➔ Achsen x, y und a Geschwindigkeit 10.0 [U/s], Achse z Geschwindigkeit 0 und Joystick-Betrieb „EIN„.
	!speed y 25	➔ Achse y Geschwindigkeit 25 und Joystick-Betrieb „EIN„.
	!speed y -25	➔ Achse y Geschwindigkeit 25 in negative Richtung Joystick-Betrieb „EIN„.
	?speed	➔ Auslesen der eingestellten Geschwindigkeiten.
	?speed y	➔ Auslesen der eingestellten Geschwindigkeit von Achse y.
Rückmeldung:	Aktuelle Geschwindigkeiten	
Fehlercode:	--	
Beispiel:	!speed 33 11 (Achsen x Geschwindigkeit 33.0 [U/s], Achse y Geschwindigkeit 11.0 [U/s] und Joystick-Betrieb „EIN„) ?speed	
Anmerkung:	Will man nach dem Ausführen des speed-Befehls wieder absolut oder relativ positionieren, muss man erst mit !speed 0 den digitalen Joystick ausschalten und die Geschwindigkeit neu setzen.	

Joystick-Richtung + Joystick-Sperren	
Befehl:	!joydir oder ?joydir
Parameter:	0, +-1, +-2, x, y, z, und a
Beschreibung:	Mit der Eingabe von joydir wird die Drehrichtung der Motoren, bei Auslenkung des Joysticks verändert und Achsen gesperrt oder freigegeben. !joydir –1 –1 1 1 = Bei x- und y-Achse negative, bei z- und a-Achse positive Drehrichtung !joydir –2 –2 2 2 = Wie bei obigem Beispiel, jedoch werden die Achsen, wenn sie länger als 1s nicht bewegt wurden in den Stromreduzierten Mode umgeschaltet. !joydir z 0 = z-Achse ist gesperrt. Besonderheit: Da nur ein 3-Achsen Joystick vorgesehen ist, wirkt die 3‘te Joystick-Achse auf die z- und a-Achse.
Rückmeldung:	Eingestellte Richtungen oder Zustand.
Fehlercode:	--
Beispiel:	!joydir-1 (negatives Vorzeichen bei Achse x) ?joydir

Joystick	
Befehl:	!joy oder ?joy
Parameter:	0, 1, 2, 3, 4 und 5
Bemerkung:	Der Joystickschalter muß auf Automatik stehen
Beschreibung:	!joy 0 ➔ Joystick „AUS„
	!joy 1 ➔ Joystick „EIN„ ohne Positionszählung
	!joy 2 ➔ Joystick „EIN„ mit Positionszählung
	!joy 3 ➔ Joystick „EIN„ mit Positionszählung und periodischer Positionsrückmeldung.
	!joy 4 ➔ Joystick „EIN„ mit Positionszählung (Encoderwerte, wenn vorhanden)
	!joy 5 ➔ Joystick „EIN„ mit Positionszählung und periodischer Positionsrückmeldung (Encoderwerte, wenn vorhanden).
	?joy ➔ aktueller Zustand
	?joy ➔ **M (Joystick manual über Schalter eingesschaltet)**
Rückmeldung:	Aktuelle Position oder aktueller Zustand des Joystickbetriebs. Wird der Joystick ausgeschaltet kommt als Rückmeldung für jede Achse ein @ wenn Autostatus =1
Fehlercode:	--
Beispiel:	!joy 2 (Joystick „EIN„ mit Positionszählung) ?joy (Abfrage des aktueller Zustands)

Joystick-Geschwindigkeit	
Befehl:	!joyspeed oder ?joyspeed
Parameter:	x, y, z oder a 1,2 oder 3 +- maximale Geschwindigkeit (vel)
Bemerkung:	Für Zusatz-Bedienpult
Beschreibung:	!joyspeed 0 25 ➔ Parameter 0 mit Geschwindigkeit 25 beschreiben.
	?joyspeed 1 ➔ Auslesen der eingestellten Geschwindigkeit von Parameter 1.
Rückmeldung:	Aktuelle eingestellte Geschwindigkeiten
Fehlercode:	--
Beispiel:	!joyspeed 2 11 (Parameter 2 mit Geschwindigkeit 11 beschreiben.) ?joyspeed 2

Joystick Window (joywindow)	
Befehl:	!?joywindow
Parameter:	0 – 100
Beschreibung:	Mit der Eingabe von joywindow wird der Analogbereich festgelegt in dem sich die Achsen nicht bewegen. Gilt für alle Achsen.
	joywindow 10 => Es muß eine größere Auslenkung des Joystickes vorliegen als 10 („Punkte“), damit die Achsen sich bewegen.
	?joywindow => Auslesen des Joystick - Fensters .
	Beispiel: Nullstellung Joystick = 512 (Analogwert) Joywindow = 10 D.h., daß die Achsen bei Werten < 502 und > 522 bewegt werden
Rückmeldung:	Eingestelltes Fenster
Fehlercode:	--
Beispiel:	?joywindow (Auslesen der Fenstergröße)

Joystick-Achszuordnung		
Befehl:	!joychangeaxis oder ?joychangeaxis	
Parameter:	0, 1	
Beschreibung:	!joychangeaxis 0	➔ Ändert die Zuordnung der AD-Joystickkanäle (konventionelle Joystickauswertung)
	!joychangeaxis 1	➔ Ändert die Zuordnung der AD-Joystickkanäle (X- und Y-Achszuordnung wird getauscht)
Rückmeldung:	0 oder 1	
Fehlercode:	--	
Beispiel:	! joychangeaxis 1 (Vertauschen der Zuordnung von X- und Y-Achse) ? joychangeaxis	

Konfiguration Joystick On/Off	
Befehl:	!?savejoyonoff
Parameter:	0 = Joystick ist nach dem Einschalten des PCs ausgeschaltet 1 = Joystick ist nach dem Einschalten des PCs eingeschaltet
Bemerkung:	Diesen Befehl gibt es nur bei der LSTEP-PCI
Beschreibung:	?savejoyonoff => Lesen des aktuellen Zustandes !savejoyonoff 1 => nach anschließendem Savebefehl und Reset der Steuerung oder (Neustart des PCs) ist der Joystick eingeschaltet
Rückmeldung:	0 oder 1
Fehlercode:	--
Beispiel:	!savejoyonoff 1 !save (Einstellung wird ins Flash gebrannt) !reset (NEUSTART)

Handradauswertung:

Die Bewegung des Tisches reagiert dynamisch auf die Drehung des Handrades. Bei langsamer Drehung werden Mikroschritte verfahren, und bei schnelleren Drehungen ein Geschwindigkeitsprofil. Mit den Befehlen hwvel und hwaccel lassen sich die max. Geschwindigkeit und die Beschleunigung im Handradbetrieb einstellen.

Handrad	
Befehl:	!hw oder ?hw
Parameter:	0, 1, 2, 3, 4 und 5
Bemerkung:	Alternativ zum Joystick kann ein Handrad angeschlossen werden.
Beschreibung:	!hw 0 ➔ Handrad „AUS„
	!hw 1 ➔ Handrad „EIN„ ohne Positionszählung
	!hw 2 ➔ Handrad „EIN„ mit Positionszählung
	!hw 3 ➔ Handrad „EIN„ mit Positionszählung und periodischer Positionsrückmeldung.
	!hw 4 ➔ Handrad „EIN„ mit Positionszählung (Encoderwerte, wenn vorhanden).
	!hw 5 ➔ Handrad „EIN„ mit Positionszählung und periodischer Positionsrückmeldung (Encoderwerte, wenn vorhanden).
	?hw ➔ aktueller Zustand
Rückmeldung:	Aktuelle Position oder aktueller Zustand des Handradbetriebs.
Fehlercode:	--
Beispiel:	!hw 2 (Handrad „EIN„ mit Positionszählung) ?hw (Abfrage des aktueller Zustands)

Handrad Geschwindigkeit	
Befehl:	!hwvel oder ?hwvel
Parameter:	X und Y 0.0001 bis 40.0000 U/s
Bemerkung:	Diesen Befehl gibt es nur in Verbindung mit einem Handrad
Beschreibung:	!hwvel 1 1 Die max. erreichbare Geschwindigkeit für X + Y ist 1U/s
Rückmeldung:	Wert der eingestellten Geschwindigkeit
Fehlercode:	--
Beispiel:	!hwvel 0.5 0.5 Die Achsen X und Y fahren mit max. 0,5 U/s ?hwvel

Handrad Beschleunigung	
Befehl:	!hwaccel oder ?hwaccel
Parameter:	X und Y 0 – max. Beschleunigung
Bemerkung:	Diesen Befehl gibt es nur in Verbindung mit einem Handrad
Beschreibung:	!hwaccel 0.5 0.5 Die Beschleunigung für X + Y ist 0.5m/s²
Rückmeldung:	Wert der eingestellten Beschleunigung
Fehlercode:	--
Beispiel:	!hwaccel 1 1 Die Beschleunigung für X + Y ist 1m/s² ?hwaccel

Bedienpult	
Befehl:	!bpz oder ?bpz
Parameter:	0,1 oder 2
Bemerkung:	Für Zusatz-Bedienpult mit Trackball
Beschreibung:	!bpz 0 => Bedienpult „AUS“
	!bpz 1 => Bedienpult aktivieren und Trackball mit 0,1µ Schrittauflösung betreiben, Joyspeedtasten aktiv
	!bpz 2 => Bedienpult aktivieren und Trackball mit Faktor betreiben, Joyspeedtasten aktiv.
	!bpz 3 => Bedienpult aktivieren und Trackball mit 0,1µ Schrittauflösung betreiben, Funktionstasten aktiv
	!bpz 4 => Bedienpult aktivieren und Trackball mit Faktor betreiben, Funktionstasten
	?bpz => Auslesen des eingestellten Zustands
Rückmeldung:	Aktueller Zustand
Fehlercode:	--
Beispiel:	!bpz 1 (Bedienpult aktivieren und Trackball mit 0,1µ Schrittauflösung betreiben)

Bedienpult Trackball Factor	
Befehl:	!bpztf oder ?bpztf
Parameter:	0,01 bis 10,00
Bemerkung:	Für Zusatz-Bedienpult
Beschreibung:	!bpztf 1 => Trackball – Factor = 1, d.h. Ein Trackball-Impuls ergibt ein Motor-Increment.
	?bpztf => Auslesen des eingestellten Factors
Rückmeldung:	Aktueller Factor
Fehlercode:	--
Beispiel:	?bpztf => Auslesen des eingestellten Factors

Bedienpult Trackball Back Lash	
Befehl:	!bpzbl oder ?bpzbl
Parameter:	0,0001 bis 0,015 mm
Bemerkung:	Für Zusatz-Bedienpult
Beschreibung:	!bpzbl 0.01 0.005 => Umkehrspiel von x-Achse = 10µm und y-Achse = 5µm.
	!bpzbl z 0.001 => Umkehrspiel von z-Achse = 1µm.
	?bpzbl => Auslesen der eingestellten Lose
Rückmeldung:	Aktueller Lose
Fehlercode:	--
Beispiel:	?bpzbl => Auslesen des eingestellten Lose

4.11 Ein/Ausgänge *(nicht bei ECO-STEP)*

Optional kann die LSTEP mit einer Zusatzkarte ausgerüstet werden, mit der je 16 Schaltein/Ausgänge sowie zwei Analog- Ausgänge verfügbar werden. Die Zusatzkarte für die LSTEP-PCI verfügt nur über 16 Schaltein/Ausgänge.
Zur Benutzung dieser Ein/Ausgänge muß die LSTEP in der entsprechenden Ausführung bestellt werden. Am Multifunktionssport (Kapitel 5.1) stehen auch analoge Ein/Ausgänge zur Verfügung.

Digitaler Eingang	
Befehl:	?digin
Parameter:	0 bis 15
Beschreibung:	?digin ➔ Lesen aller Inputpins
	?digin 8 ➔ Lesen des Inputpins 8
Rückmeldung:	Zustand der Inputpins
Fehlercode:	--
Beispiel:	?digin (Lesen aller Inputpins)

Digitaler Ausgang		
Befehl:	!digout oder ?digout	
Parameter:	0 bis 15	
Beschreibung:	!digout 11110000	➔ Es werden die Outputpins 0,1,2,3 auf „1„ und die Outputpins 4,5,6,7 auf „0„ gesetzt.
	!digout 5 1	➔ Es wird Outputpin 5 auf „1„ gesetzt.
	?digout	➔ Lesen des aktuellen Zustands aller Outputpins.
	?digout 8	➔ Lesen des aktuellen Zustands von Outputpin 8
Rückmeldung:	Zustand der Outputpins	
Fehlercode:	--	
Beispiel:	!digout 7 0 (Setze Outputpin 7 auf „0„) ?digout (Lesen aller Outputpins)	

Funktion der digitalen Ein-/Ausgänge	
Befehl:	!digfkt oder ?digfkt
Parameter:	0 bis 15 (Input/Output), 16 (alle 16 Portpins) 0 bis 4 (Funktion) 0 bis 100 mm (Strecke) oder Polarität der Eingänge x, y, z und a (Achse)
Beschreibung:	**Funktionen:**
	0 ➔ Funktion wird ausgeschaltet
	1 ➔ Zuordnung des Not_Stop-Pins.
	2 ➔ Aktivierung eines Ausgang in Abhängigkeit der eingestellten Strecke vor der Zielposition.
	3 ➔ Aktivierung eines Ausgang in Abhängigkeit der eingestellten Strecke nach der Startposition.
	4 ➔ 2&3
	Befehle:
	!digfkt 7 2 78.9 z ➔ Ausgang 7 wird 78.9mm vor erreichen der Zielposition aktiviert.
	!digfkt 14 1 ➔ Eingang 14 wird als Not_Stop benutzt.
	!digfkt 16 0 ➔ Alle Funktionen werden auf 0 gesetzt.
	!digfkt 16 0 0 ➔ Alle Eingänge high aktiv
	!digfkt 16 0 1 ➔ Alle Eingänge low aktiv
	!digfkt 5 0 0 ➔ Eingang 5 high aktiv
	?digfkt 16 oder ?digfkt ➔ Es werden die aktuellen Funktionszustände aller Ein- und Ausgänge angezeigt.
	?digfkt 6 Es werden die aktuellen Funktionszustände von Eingang 6 und Ausgang 6 angezeigt.
	?digfkt 7 4 Es wird die zugehörige Strecke und Achszuordnung angezeigt.
Rückmeldung:	Alle Einstellungen
Fehlercode:	--
Beispiel:	!digfkt 7 0 (Setze die Funktion von Ein- und Ausgang 7 auf „0„) ?digfkt 9 (Lesen der Funktion von Ein- und Ausgang 9)

EXTENDED DIGITAL INPUT	
Befehl:	?edigin
Parameter:	0 bis 15
Bemerkung:	Diesen Befehl gibt es nur bei LS44-Steuerungen.
Beschreibung:	?edigin = Lesen des aktuellen Zustands aller zusätzlichen Inputpins ?edigin 8 = Lesen des aktuellen Zustands des zusätzlichen Inputpins 8
Rückmeldung:	Zustand der zusätzlichen Inputpins
Fehlercode:	--
Beispiel:	?edigin (Lesen aller zuätzlichen Inputpins)

EXTENDED DIGITAL OUTPUT	
Befehl:	!edigout oder ?edigout
Parameter:	0 bis 15
Bemerkung:	Diesen Befehl gibt es nur bei LS44-Steuerungen
Beschreibung:	!edigout 11110000 = Es werden die zusätzlichen Outputpins 0,1,2,3 auf „1“ und 4,5,6,7 auf „0“ gesetzt. !edigout 5 1 = Es wird der zusätzliche Outputpin 5 auf „1“ gesetzt. ?edigout = Lesen des aktuellen Zustands aller zusätzlichen Outputpins ?edigout 8 = Lesen des aktuellen Zustands des zusätzlichen Outputpins 8
Rückmeldung:	Zustand der zusätzlichen Outputpins
Fehlercode:	--
Beispiel:	!edigout 7 0 (Setze den zusätzlichen Outputpin 7 auf „0“) ?edigout (Lesen aller zusätzlichen Outputpins)

FUNKTION der zusätzlichen digitalen Ein-und Ausgänge	
Befehl:	!edigfkt oder ?edigfkt
Parameter:	0 bis 15 (Input/Output), 16 (alle 16 Portpins) 0 (Funktion)
Bemerkung:	Diesen Befehl gibt es nur bei LS44-Steuerungen.
Beschreibung:	Funktion: 0 = Keine Beeinflussung der Ein-/Ausgänge und Einstellung der Polarität (0 = High-, 1 = Low-Active) !edigfkt 16 0 = Alle Funktionen werden auf 0 gesetzt. ?edigfkt 16 oder ?edigfkt = Es werden die aktuellen Funktionszustände aller zusätzlichen Ein- und Ausgänge angezeigt ?edigfkt 6 = Es werden die aktuellen Funktionszustände des zusätzlichen Eingangs/Ausgangs 6 angezeigt
Rückmeldung:	Alle Einstellungen
Fehlercode:	--
Beispiel:	!edigfkt 7 0 (Setze die Funktion des zusätzlichen Ein- und Ausgangs 7 auf „0“) ?digfkt 9 (Lesen der Funktion des zusätzlichen Ein- und Ausgangs 9)

Auf der Zusatz-Karte für die LSTEP sind zusätzlich zwei analoge Ausgänge vorhanden. Die Ausgänge sind standardmäßig für 0...5V ausgelegt. Andere Ausgangsspannungsbereiche (z.B. +/- 5V, +/- 10V, 0...10V,...) sind auf Anfrage möglich. Die Ausgänge sind mit +/- 5mA belastbar. Der Innenwiderstand beträgt ca. 100 Ohm.

Analoger Ausgang		
Befehl:	!anaout oder ?anaout	
Parameter:	0 bis 100 % 0, 1 und 2 (Analogkanäle) c (c = channel)	
Bemerkung:	Channel 0 und 1 sind auf der Zusatz I/O-Karte der LSTEP. Channel 2 ist auf dem Multifunktionsport der LSTEP. Für die PCI-Karte gibt es nur einen analogen Ausgang, es ist Channel 0 und liegt auf dem Multifunktionsport.	
Beschreibung:	!anaout 100 50	➔ Es wird der erste Analogkanal auf 100% (volle Spg.) und der zweite auf 50% (halbe Spg.) gestellt.
	!anaout c 1 25	➔ Es wird Analogkanal 1 auf 25% eingestellt.
	?anaout	➔ Lesen des aktuellen Zustands aller Analogkanäle.
	?anaout c 2	➔ Lesen des aktuellen Zustands von Analogkanal 2.
Rückmeldung:	Zustand der prozentualen Aussteuerung der Analogkanäle.	
Fehlercode:	--	
Beispiel:	!anaout c 1 0 (Setze Analogkanal 1 auf „0„) ?anaout (Lesen aller Analogkanäle)	

ANALOG INPUT	
Befehl:	?anain
Parameter:	0 bis 10 (Analogkanäle) c (c = channel)
Beschreibung:	?anain c 2 => Lesen des aktuellen Zustands von Analogkanal 2
Rückmeldung:	Zustand je nach Analogkanal
Fehlercode:	--
Beispiel:	?anain c 2 (Lesen des aktuellen Zustands von Analogkanal 2)

Channel						
0	MFP	Pin 24	Joystick X			
1	MFP	Pin 12	Joystick Y			
2	MFP	Pin 25	Joystick Z			
3	MFP	Pin 26	ST11 (nicht auf 25pol DSub)			
4	Speedpoti bei LSTEP mit Anzeige					
5	Motorspannung bei der LSTEP-PCI					
6	MFP	Pin 8	oder	LSTEP-PCI	St10	Pin 1
7	MFP	Pin 20	oder	LSTEP-PCI	St10	Pin 2
8	MFP	Pin 7	oder	LSTEP-PCI	St10	Pin 3
9	MFP	Pin 19	oder	LSTEP-PCI	St10	Pin 4
10	MFP	Pin 6	oder	LSTEP-PCI	St10	Pin 6/7

4.12 Auswertung von inkrementalen Meßsystemen *(nicht bei ECO-STEP)*

An der Steuerung können gleichzeitig Achsen mit und ohne Geber betrieben werden. Hierzu überprüft die Steuerung während des Kalibriervorganges, ob Geber angeschlossen sind, sofern diese durch *encmask* freigegeben wurden. Das Ergebnis dieser Überprüfung kann mit dem Befehl *?enc* überprüft werden. Die Steuerung unterscheidet hierbei allerdings nicht zwischen falsch angeschlossenem und fehlendem Geber.
Auch beim Kalibrieren auf Referenzmarke, fährt die Achse in negative Richtung in den Null-Endschalter, macht eine Richtungsumkehr und fährt mit der Geschwindigkeit die über calrefspeed eingestellt wurde bis zur Referenzmarke. Hat ein System keine Endschalter (z.B. eine Drehachse) wird direkt auf die Referenzmarke kalibriert, wenn vorher alle Endschalter deaktiviert wurden.
Ab der Firmware Version „T03.19.06-2001“ sind die Sin.- Cos.- Signale nur noch passend in ihrer Zählrichtung zur Motorzählrichtung anzuschließen. Ein Abgleich zur Encoder-Referenzmarke ist nicht mehr notwendig.

Gebermaske für benutzte Geber	
Befehl:	!encmask oder ?encmask
Parameter:	X, y, z und a 0,1 (On,Off)
Bemerkung:	Freigabe der einzelnen Geber.
Beschreibung:	!encmask 1 0 1 ➔ x- und z-Geber aktiv, y-Geber deaktiviert.
	?encmask ➔ Die Gebermaske aller Geber wird angezeigt.
	?encmask x ➔ Anzeigen der Gebermaske von Achse x.
Rückmeldung:	Geberzmaske
Fehlercode:	--
Beispiel:	!encmask 1 0 (Geber x-Achse freigegeben, Geber y-Achse nicht freigegeben) ?encmask

Gebermaske für erkannte Geber	
Befehl:	?enc
Parameter:	X, y, z und a 0 oder 1 (On,Off)
Bemerkung:	Wenn Geber aktiviert werden, die nicht vorhanden sind kann es zu Fehlfunktionen kommen.
Beschreibung:	!enc 1 0 1 ➔ x- und z-Geber aktiv, y-Geber deaktiviert.
	?enc ➔ Alle Geberzustände werdenb angezeigt.
	?enc x ➔ Anzeigen der Gebermaske von Achse x.
Rückmeldung:	Geberzustand
Fehlercode:	--
Beispiel:	!enc 1 0 (Geber x-Achse aktiv, Geber y-Achse deaktiviert) ?enc

Signalperiode / Linear Encoder	
Befehl:	!encperiod oder ?encperiod
Parameter:	X, y, z und a 0.0001 – Spindelsteigung * 0.8 (mm)
Beschreibung:	!encperiod 0.5 0.020 ➔ Periodenlänge des Gebersignals bei x-Achse ist 500µm und bei y-Achse ist 20µm.
	?encperiod ➔ Alle Geberperiodenlängen werden angezeigt.
	?encperiod x ➔ Anzeige der Geberperiodenlänge von Achse x.
Rückmeldung:	Geberperiodenlänge in mm
Fehlercode:	--
Beispiel:	!encperiod 0.1 (Geberperiodenlänge der x-Achse ist 0.1mm) ?encperiod

Encoder Resolution / Rotary Encoder	
Befehl:	!?encres
Parameter:	1 bis 40000
Bemerkung:	Gibt die Anzahl der Encodersignalperioden pro Motorumdrehung an. Ist der Encoder hinter einem Getriebe montiert, sollte das Verhältnis von Perioden zum Getriebefaktor eine ganze Zahl ergeben.
Beschreibung:	?encres !encres 250 500 1000
Rückmeldung:	-
Fehlercode:	--
Beispiel:	!encres 500 500 500 Bei den Achsen X,Y,Z werden 500 Signalperioden pro Motorumdrehung an die Steuerung übergeben.

Geber-Referenzsignal		
Befehl:	!encref oder ?encref	
Parameter:	X, y, z oder a 0 oder 1	
Beschreibung:	!encref 1 1 0	➔ Beim Kalibrieren wird das Referenzsignal der Geber x und y ausgewertet.
	!encref z 1	➔ Beim Kalibrieren wird das Referenzsignal des Gebers der z-Achse ausgewertet.
	?encref	➔ Die aktuelle Einstellung wird angezeigt.
	?encref y	➔ Die aktuelle Einstellung der y-Achse wird angezeigt.
Rückmeldung:	0 oder 1	
Fehlercode:	--	
Beispiel:	!encref 0 (Keine Referenzsignalauswertung der x-Achse) ?encref	

Geberwertanzeige		
Befehl:	!encpos oder ?encpos	
Parameter:		
Beschreibung:	!encpos 1	➔ Bei der Positionsabfrage werden die Geberwerte der erkannten Geber angezeigt.
	?encpos	➔ Die aktuelle Einstellung wird angezeigt.
Rückmeldung:	0 oder 1	
Fehlercode:	--	
Beispiel:	!encpos 0 (Geberpositionsanzeige „AUS„) ?encpos	

Geber-Fehler	
Befehl:	!encerr oder ?encerr
Parameter:	X, y, z, a 0
Beschreibung:	!encerr 0 0 0 ➔ Clear Geberfehler-Meldungen von x-, y- und z-Achse.
	!encerr a 0 ➔ Clear Geberfehler-Meldung von a-Achse.
	?encerr ➔ Die aktuellen Geberfehler-Meldungen aller Achsenwerden angezeigt.
	?encerr z ➔ Die aktuelle Geberfehler-Meldung der z-Achse wird angezeigt.
Rückmeldung:	0 oder e
Fehlercode:	--
Beispiel:	!encerr 0 (Clear Geberfehler-Meldung von a-Achse) ?encerr

Geber – Position PCI (EncoderReadPositionPCI)	
Befehl:	?hwcount
Parameter:	X, y, z, a
Beschreibung:	?hwcount => Lese alle Geberpositionen ?hwcount a => Lese Geberposition der A-Achse
Rückmeldung:	Zählerwert 4-fach interpoliert
Fehlercode:	--
Beispiel:	?hwcount x (Lese Geberposition der a-Achse)

Geber – Position PCI (EncoderClear-PositionPCI)	
Befehl:	!clearhwcount
Parameter:	X, y, z, a
Beschreibung:	!clearhwcount => Clear alle Geber – Zähler. !clearhwcount a => Clear Geber – Zähler der A-Achse
Rückmeldung:	--
Fehlercode:	--
Beispiel:	!clearhwcount x (Clear Geber – Zähler der A-Achse)

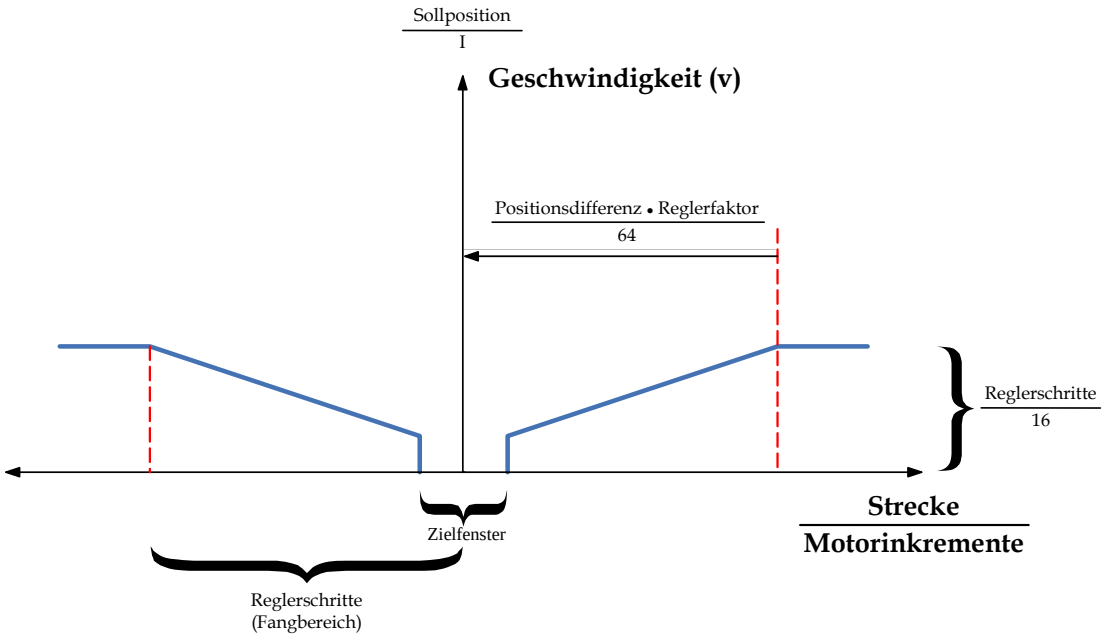
4.13 Reglereinstellungen für LSTEP *(nicht bei ECO-STEP)*

Mit Hilfe verschiedener Parameter kann das Regelverhalten im closed Loop Betrieb beeinflußt werden.

Diese Parameter sind:

1. Ctrc (Regleraufruf)
2. Ctrs (Reglerschritte / Fangbereich)
3. Ctrf (Reglerfaktor)
4. Ctrd (Reglerverzögerung)
5. Ctrt (Reglerüberwachung / Timeout)

Die Werte für **Ctrc, Ctrd** und **Ctrt** gelten für alle Achsen gleichzeitig. Die Werte für **Ctrs** und **Ctrf** sind für jede Achse einzeln einstellbar.



Die Positionsdifferenz ist die Abweichung von der momentanen Istposition zur vorgegebenen Zielposition. Liegt die Istposition ausserhalb des eingestellten Fangbereiches, so verfährt die Steuerung mit konstanter Geschwindigkeit (wenn "ctrm 0" gesetzt ist). Diese wird über **Ctrs** eingestellt. Innerhalb des Fangbereiches wird die Verfahrgeschwindigkeit der Positionsdifferenz angepaßt. Diese Anpassung kann über den Parameter **Ctrf** beeinflußt werden.

Die Bedeutung der Parameter ist:

- Ctrc Der Wert in *Ctrc* gibt die Abtastzeit an, mit der der Regler aufgerufen wird. In der Regel gilt, daß die Dämpfung des Gesamtsystems mit steigender Abtastzeit erhöht wird.
- Ctrs Der Inhalt von *Ctrs* entspricht in Abhängigkeit von der Dimension einer Strecke, die den Fangbereich für die jeweilige Achse angibt.

  Beispiel: Ctrs = 500, Dimension = 1 ➔ 500 • 1µm = 500µm)

  Ist die Positionsdifferenz größer als *Ctrs*, wird sich der Zielposition mit einer konstanten Geschwindigkeit genähert.
- Ctrf Innerhalb des Fangbereiches wird durch eine mathematische Funktion für jede Achse einzeln die Positionsdifferenz manipuliert. Der Reglerfaktor gibt an wie stark sich die jeweilige Positionsdifferenz auf die Geschwindigkeit, mit der die Zielposition angefahren wird, auswirkt.
- Ctrd *Ctrd* gibt an wie lange die ausgewählten Achsen das Zielfenster nicht verlassen dürfen, damit die Meldung Position erreicht gesendet wird.
- Ctrt Der Regler- Timeout begrenzt die Zeit, die der Regler zur Verfügung hat, um eine Positionsdifferenz auszuregeln.

Beispiel (ctrs):

$$\frac{\text{1mm Spindelsteigung}}{\text{50000 Schritte pro Motorumdrehung}} = 0{,}02\ \mu\text{m (= ein Motorinkrement)}$$

$$\frac{\text{Fangbereich = 0,1 mm (=100 µm)}}{\text{0,02 µm (Motorinkrement)}} = 5000\ \text{Motorinkremente}$$

$$\frac{\text{5000 Motorinkremente}}{\text{16}} = 312{,}5\ \text{Motorinkremente}$$

$$\frac{\text{312,5 Motorinkremente}}{\text{ctrc (Regleraufruf)}} = V_{Konstant}$$

Zielfenster	
Befehl:	!twi oder ?twi
Parameter:	X, y, z und a 1 bis 25000 (Motorinkremente) 0.1 bis Spindelsteigung/2 (µm) 0.0001 bis Spindelsteigung/2 (mm)
Bemerkung:	Ein- und Ausgabewerte sind abhängig von der Dimension.
Beschreibung:	!twi 1.0 0.002 ➔ Bei Achse x beträgt das Zielfenster 1mm und bei der y-Achse 2µm (bei Dim = 2). Die anderen Achsen bleiben unverändert.
	!twi z 0.1 ➔ Bei der z-Achse wird das Zielfenster auf 0.1µm (bei Dim = 1) eingestellt.
	?twi ➔ Alle eingestellten Zielfenster werden angezeigt.
	?twi x ➔ Anzeigen des eingestellten Zielfenster von Achse x.
Rückmeldung:	Wirklich eingestellte Zielfenster (Rundungsfehler werden angezeigt)
Fehlercode:	--
Beispiel:	!twi 10 (Die x-Achse hat ein Zielfenster von 10 Motorincrementen (bei Dim = 0)). ?twi

Regler	
Befehl:	!ctr oder ?ctr
Parameter:	X, y, z und a
	0 ➔ Regler „AUS„
	1 ➔ Regler „AUS nach erreichen der Zielposition„
	2 ➔ Regler „Immer EIN„
	3 ➔ Regler „AUS nach erreichen der Zielposition„ mit reduziertem Strom.
	4 ➔ Regler „Immer EIN„ mit reduziertem Strom.
Beschreibung:	!ctr y 2 ➔ Regler der y-Achse „Immer EIN„.
	?ctr ➔ Alle Reglerzustände werden angezeigt
	?ctr x ➔ Anzeige des Reglerzustandes von Achse x
Rückmeldung:	Reglerzustände
Fehlercode:	--
Beispiel:	!ctr 0 0 0 0 (Alle Regler „AUS„) ?ctr

Reglerüberwachung (Timeout)	
Befehl:	!ctrt oder ?ctrt
Parameter:	0 – 10000 (ms)
Beschreibung:	!ctrt 2 ➔ Reglerüberwachung 2ms.
	?ctrt ➔ Anzeige der Reglerüberwachung
Rückmeldung:	Reglerüberwachung
Fehlercode:	--
Beispiel:	!ctrt 0 (Reglerüberwachung „AUS„) ?ctrt

Regleraufruf	
Befehl:	!ctrc oder ?ctrc
Parameter:	1 – 100 (ms) sollte nicht kleiner 3ms eingestellt werden
Beschreibung:	!ctrc 5 ➔ Regleraufruf alle 2ms.
	?ctrc ➔ Regleraufrufzeit wird angezeigt.
Rückmeldung:	Regleraufrufzeit
Fehlercode:	--
Beispiel:	!ctrc 10 (Regleraufruf alle 10ms) ?ctrc

Regler-Schritte	
Befehl:	!ctrs oder ?ctrs
Parameter:	X, y, z und a 1 bis Spindelsteigung
Bemerkung:	Ein- und Ausgabewerte sind abhängig von der Dimension
Beschreibung:	!ctrs y 2 ➔ 2mm Reglerschritte der y-Achse (bei DIM = 2).
	?ctrs ➔ Alle Reglerschritte werden angezeigt.
	?ctrs x ➔ Anzeige des Reglerschritte von Achse x.
Rückmeldung:	Reglerschritte
Fehlercode:	--
Beispiel:	!ctrs 4 5 7 9 (Reglerschritte für alle Achsen in abhängigkeit der Dimension) ?ctrs

Regelfaktor	
Befehl:	!ctrf oder ?ctrf
Parameter:	X, y, z und a 1 – 64
Beschreibung:	!ctrf y 2 ➔ Reglerfaktor der y-Achse 2.
	?ctrf ➔ Alle Reglerfaktoren werden angezeigt.
	?ctrf x ➔ Anzeige des Reglerfaktors von Achse x.
Rückmeldung:	Reglerfaktoren
Fehlercode:	--
Beispiel:	!ctrf 1 2 3 4 (Setze alle Reglerfaktoren) ?ctrf

Reglerverzögerung	
Befehl:	!ctrd oder ?ctrd
Parameter:	0 – 100 (ms)
Beschreibung:	!ctrd y 2 ➔ Reglerverzögerung der y-Achse 2ms.
	?ctrd ➔ Alle Reglerverzögerungen werden angezeigt.
	?ctrd x ➔ Anzeige der Reglerverzögerung von Achse x.
Rückmeldung:	Reglerverzögerung
Fehlercode:	--
Beispiel:	!ctrd 0 0 0 0 (Alle Reglerverzögerungen „AUS„) ?ctrd

Regelung (Fast Move)	
Befehl:	!?ctrfm
Parameter:	0 oder 1
Bemerkung:	Bedeutung der Fast Move Funktion: Bei einer Reglerdifferenz, die größer als der Fangbereich ist wird ein neuer Vektor gestartet.
Beschreibung:	!ctrfm 1 => Fast Move Funktion wird aktiviert ?ctrfm => Anzeige des Zustandes der Fast Move Funktion
Rückmeldung:	0 = Fast Move Funktion nicht aktive 1 = Fast Move Funktion aktive
Fehlercode:	--
Beispiel:	!ctrfm 0 (Fast Move Funktion „AUS“)

Regelung (Fast Move Counter)	
Befehl:	!?ctrfmc
Parameter:	0 bis 255
Bemerkung:	Bedeutung der Fast Move Counter Funktion: Bei einer Reglerdifferenz, die größer als der Fangbereich ist wird ein neuer Vektor gestartet und der dazu gehörige Counter um eins erhöht.
Beschreibung:	!ctrfmc 0 => Clear Fast Move Counter ?ctrfmc => Anzeige der Anzahl, ausgeführter Fast Move Funktionen.
Rückmeldung:	0 bis 255
Fehlercode:	--
Beispiel:	!ctrfmc 0 (Clear Counter)

4.14 Spezielle Befehle für das MR-System

MROffset	
Befehl:	!mro oder ?mro
Parameter:	X, y, z und a +- 2048
Beschreibung:	!mro 20 –3 56 ➔ Offset sinx = 20, Offset cosx = -3 und Offset siny = 56 Punkte.
	!mro y 2 9 ➔ Offset siny = 2 und Offset cosy = 9 Punkte.
	?mro ➔ Alle Offsetwerte werden angezeigt
	?mro x ➔ Anzeige der Offsetwerte von Achse x
Rückmeldung:	Immer sin cos für jede Achse
Fehlercode:	--
Beispiel:	!mro 0 0 0 0 (Die Offsetwerte von x- und y-Achse werden auf 0 gesetzt) ?mro

Maximale Signalwerte	
Befehl:	!mrp oder ?mrp
Parameter:	X, y, z und a +- 2048
Beschreibung:	!mrp ➔ Fehler 2 (Es wären bis zu 16 Werte).
	!mrp y 1000 –1000 1000 ➔ Pos. Spitzenwert siny = 1000, neg. Spitzenwert siny = -1000 und Pos. Spitzenwert cosy = 1000 Punkte.
	?mrp ➔ Alle Spitzenwerte werden angezeigt.
	?mrp x ➔ Anzeige der Spitzenwerte von Achse x.
Rückmeldung:	Immer pos. sin, neg. sin und pos. cos, neg. cos für jede Achse.
Fehlercode:	--
Beispiel:	!mrp 0 0 0 0 (Die Spitzenwerte von der x-Achse werden auf 0 gesetzt) ?mrp

Aktuelle Signalwerte	
Befehl:	!mrt oder ?mrt
Parameter:	X, y, z und a
Beschreibung:	!mrt ➔ Fehler 2
	!mrt z ➔ Fehler 2
	?mrt ➔ Fehler 2
	?mrt x ➔ Anzeige der aktuellen Signalwerte von Achse x
Rückmeldung:	Immer 10 * (sin, cos für die jeweilige Achse)
Fehlercode:	--
Beispiel:	?mrt a (Anzeige der aktuellen Signalwerte von Achse a)

Verstärkungsfaktor		
Befehl:	!mra oder ?mra	
Parameter:	X, y, z und a 0.01 – 2.00	
Bemerkung:	Der Verstärkungsfaktor bezieht sich immer auf das Cosinus-Signal	
Beschreibung:	!mra 1 1.01 0.98	➔ Verstärkungsfaktoren für cosx = 1, cosy = 1.01 und cosz = 0.98
	!mra z 1.23	➔ Verstärkungsfaktor cosz = 1.23
	?mra	➔ Anzeige der Verstärkungsfaktoren aller Achsen.
	?mra x	➔ Anzeige des aktuellen Verstärkungs-faktors von Achse x.
Rückmeldung:	Verstärkungsfaktor	
Fehlercode:	--	
Beispiel:	!mra 1.11 (Verstärkungsfaktor der X-Achse = 1.11) ?mra a (Anzeige des aktuellen Verstärkungsfaktors von Achse a)	

Signalform	
Befehl:	!mrs oder ?mrs
Parameter:	X, y, z und a 0 oder 1
Bemerkung:	0 = Sinus und 1 = Cosinus
Beschreibung:	!mrs ➔ Fehler 2
	!mrs z 1 ➔ Auswahl des Cosinus-Signals von Achse z.
	?mrs ➔ Anzeige der Achsen Kennung und der Signalwerte.
	?mrs x ➔ Fehler 2
Rückmeldung:	Signalkennung (y 0: Werte ->)
Fehlercode:	--
Beispiel:	!mrs x 0 (Auswahl des Sinus-Signals von Achse x) ?mrs (Anzeige der Signalwerte, der Voreingestellten Achse und Signalkennung)

4.15 Auswertung von Takt und Drehrichtungsvorgaben *(nicht bei ECO-STEP)*

Optional können Achsen statt über Vektorbefehle oder Joystick auch mit Taktsignalen in Abhängigkeit des Drehrichtungssignales vor oder zurück verfahren werden. Dieser Betrieb ist auch asynchron zu Verfahrvorgängen möglich, die über Fahrbefehle ausgelöst worden sind. Hierzu steht der Multifunktionsport MFP zur Verfügung.

Hinweis:	Wie unter Takt Vor/Rück (Interne Steuerung) beschrieben kann man auch die gleiche Funktion über die serielle Schnittstelle übergeben.

4.15.1 Verfahrbereichsüberwachung

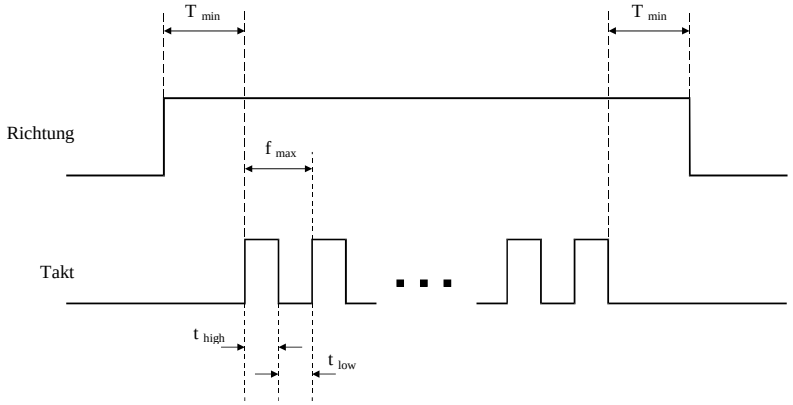
Im TVR-Betrieb wird ebenfalls überwacht, ob die zulässigen Verfahrgrenzen nicht überschritten werden. Die Verfahrgrenzen können dabei entweder über die Kombination *‚Kalibrieren'* und *‚Hub-Messen'* ermittelt worden sein. Eine andere Möglichkeit besteht darin, die Verfahrgrenzen per Befehl zu setzen.
Stellt die Steuerung fest, daß durch die aufgelaufenen Zählimpulse eine Verfahrgrenze überschritten werden würde, wird jede weitere Bewegung der Achse in dieser Richtung unterbunden. Ein Verfahren in der entgegengesetzten Richtung ist jedoch weiter möglich. Eine Meldung an den PC erfolgt nicht.

Hinweis:	Das Anwenderprogramm ist dafür verantwortlich, daß maximale Start/Stop Frequenzen des Antriebs nicht überschritten, und die jeweilige Achse beschleunigungsmäßig nicht überlastet wird.

4.15.2 Zeitliche Randbedingungen für die Signale

Die zeitliche Abfolge der Flanken von Takt- und Drehrichtungssignalen einer Achse unterliegt folgenden Randbedingungen

- frühestens T_{min} nach jedem Polaritätswechsel des Drehrichtungssignals darf der nächste Taktimpuls angelegt werden.
- spätestens T_{min} vor jedem Polaritätswechsel des Richtungssignales müssen die Taktimpulse ausgesetzt werden
- T_{min} beträgt z. Zt. 50µs.
- die maximale Frequenz der Taktimpulse darf den Wert von f_{max} = 833 kHz nicht überschreiten. Dabei müssen die Mindestzeiten T_{low} = 600ns und T_{high} = 600ns eingehalten werden.
- Als Schutz der Steuerungseingänge werden Eingangsfilter mit 470Ω und 220pF eingesetzt. Daher ist auf ausreichende Treiberleistung der Taktquelle zu achten.



Takt Vor / Rück	
Befehl:	!tvr oder ?tvr
Parameter:	X, y, z und a 0, 1, 2, 3, 4
Beschreibung:	**Funktionen:**
	0 ➔ Takt Vor/Rück „AUS„.
	1 ➔ Normale Takt Vor/Rück Bearbeitung.
	2 ➔ Takt Vor/Rück Bearbeitung arbeitet mit einem Faktor.
	3 ➔ Takt Vor/Rück Bearbeitung benötigt externe Freigabe über Start/Stopp-Eingänge.
	4 ➔ Kombination aus 2 & 3.
	Befehle:
	!tvr 1 1 ➔ Bei Achse x und y soll Takt Vor/Rück aktiviert werden.
	!tvr a 1 ➔ Bei Achse a soll Takt Vor/Rück aktiviert werden.
	?tvr ➔ Alle eingestellten Zustände werden angezeigt.
	?tvr z ➔ Der aktuelle Zustand z-Achse wird angezeigt.
Rückmeldung:	Zustand je nach Analogkanal
Fehlercode:	--
Beispiel:	!tvr 1 (Aktiviere Takt Vor/Rück bei der x-Achse) ?tvr

Faktor Takt Vor / Rück	
Befehl:	!tvrf oder ?tvrf
Parameter:	X, y, z und a 0.01 – 100.00
Beschreibung:	!tvrf 1.00 1.00 ➔ Bei den Achsen x und y soll Takt Vor/Rück mit dem Faktor 1 arbeiten (d.h. Ein Takt = Ein Motorincrement). !tvrf a 1 ➔ Bei Achse a soll Takt Vor/Rück mit dem Faktor 1 arbeiten ?tvrf ➔ Alle eingestellten Faktoren werden angezeigt ?tvrf z ➔ Der aktuelle Faktor z-Achse wird angezeigt
Rückmeldung:	Faktorwerte
Fehlercode:	--
Beispiel:	!tvrf 10.00 (Faktor = 10.00 bei der x-Achse) (Ein Takt = Zehn Motorinkremente) ?tvrf

Takt Vor / Rück (Interne Steuerung)	
Befehl:	Px, nx, py, ny, pz, nz, pa, na
Parameter:	Keine
Beschreibung:	Alle Befehle haben die gleiche Auswirkung wie ein externer Takt mit Richtungsinformation. Der erste Buchstabe gibt an, ob eine positive (p) oder negative (n) Bewegung ausgeführt werden soll. Der zweite Buchstabe gibt an, welche Achse bewegt werden soll.
Rückmeldung:	Keine
Fehlercode:	--
Beispiel:	py (1Takt Vorwärts bei der y-Achse)

4.15.3 Takt und Drehrichtngs-Ausgänge für zusätzliche Achsen

TVR Output	
Befehl:	!?tvrout
Parameter:	X, y, z und a 0 oder 1
Bemerkung	X, y, z und a sind zusätzliche Achsen, neben den eigentlichen Hauptachsen x, y, z und a
Beschreibung:	!tvrout 1 1 = Bei Achse x und y soll Takt Vor/Rück aktiviert werden !tvrout a 1 = Bei Achse a soll Takt Vor/Rück aktiviert werden ?tvrout = Alle eingestellten Zustände werden angezeigt ?tvrout z = Der aktuelle Zustand z-Achse wird angezeigt
Rückmeldung:	0 => Takt Vor/Rück „AUS“ 1 => Takt Vor/Rück „EIN“
Fehlercode:	--
Beispiel:	!tvrout 1 (Aktiviere Takt Vor/Rück bei der x-Achse) ?tvrout

TVR Out resolution	
Befehl:	!?tvrores
Parameter:	X, y, z und a 0 bis 51200
Bemerkung	Hier wird die Auflösung der an zu steuernden Endstufe eingegeben.
Beschreibung:	!tvrores 1000 1000 = Bei Achse x und y wird eine Auflösung von 1000 Impulsen pro Umdrehung eingestellt. ?tvrores = Alle eingestellten Auflösungen werden angezeigt
Rückmeldung:	0 bis 51200 für jede Achse
Fehlercode:	--
Beispiel:	!tvrores z 2500 (Auflösung der z-Achse beträgt 2500 I/U)

TVR Out Pitch	
Befehl:	!?tvropitch
Parameter:	X, y, z und a 0.001 – 100
Bemerkung	Diese Angabe ist erforderlich, damit eine Korrekte Bewegung ausgeführt werden kann.
Beschreibung:	!tvropitch 1 1 = Bei Achse x und y wird eine Spindel mit 1mm Steigung eingesetzt. ?tvropitch = Alle eingestellten Spindelsteigungen werden angezeigt
Rückmeldung:	0.001 bis 100
Fehlercode:	--
Beispiel:	!tvropitch y 4 (Die Spindelsteigung für die y-Achse beträgt 4mm)

TVR Out acceleration	
Befehl:	!?tvroa
Parameter:	X, y, z und a 0.01 - 1500 U/s² beschleunigt
Beschreibung:	?tvroa = Alle eingestellten Beschleunigungen werden angezeigt !tvroa 100 100 = Die X- und Y-Achse werden mit 100 U/s² beschleunigt
Rückmeldung:	0.01 bis 1500 [U/s²]
Fehlercode:	--
Beispiel:	!tvroa z 50 (Die z-Achse wird mit 50 U/s² beschleunigt)

TVR Out velocity	
Befehl:	!?tvrov
Parameter:	X, y, z und a 0 bis 40.0 Umdrehung pro Sekunde
Beschreibung:	!tvrov a 10 => Bei Achse a soll mit max. 10 U/s betrieben werden ?tvrov = Alle eingestellten Geschwindigkeiten werden angezeigt
Rückmeldung:	0 bis 40.0 [U/s] für jede Achse
Fehlercode:	--
Beispiel:	!tvrov 1 (Die x-Achse soll mit einer max. Geschwindigkeit von 1 U/s betrieben werden)

TVR Out position	
Befehl:	!?tvropos
Parameter:	X, y, z und a Min. Bereichsgrenze bis max. Bereichsgrenze
Bemerkung:	Siehe !?pos
Beschreibung:	tvropos 45 88 => Setze Position von x- und y-Achse ?tvropos = Alle aktuellen Positionswerte werden angezeigt
Rückmeldung:	Positionswert (in Abhängigkeit der Dimension)
Fehlercode:	--
Beispiel:	!tvropos 1 (Setze die Position der x-Achse)

TVR Out move absolute	
Befehl:	tvromoa
Parameter:	X, y, z und a +- Verfahrbereich
Bemerkung:	Siehe moa ! Die Eingabe ist auf max. 3 Nachkommastellen begrenzt. Die Eingabe ist abhängig von der Dimension.
Beschreibung:	tvromoa 1 1 = Die Achsen x und y werden auf die Position 1 verfahren
Rückmeldung:	--
Fehlercode:	--
Beispiel:	!tvromoa z 3.5 (Positioniere die z-Achse auf die Position 3.5)

TVR Out move relative	
Befehl:	tvromor
Parameter:	X, y, z und a +- Verfahrbereich
Bemerkung:	Siehe mor ! Die Eingabe ist auf max. 3 Nachkommastellen begrenzt. Die Eingabe ist abhängig von der Dimension.
Beschreibung:	tvromor 1 1 = Die Achsen x und y werden um 1mm (Dim = 2) verfahren
Rückmeldung:	--
Fehlercode:	--
Beispiel:	!tvromor z 3.5 (Bewege die z-Achse um 3.5mm (Dim = 2))

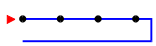
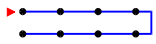
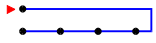
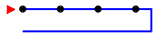
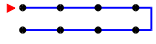
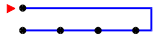
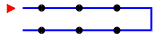
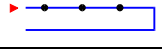
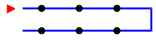
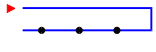
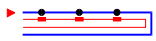
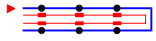
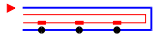
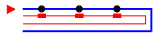
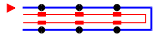
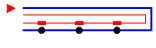
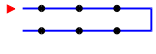
TVR Out status	
Befehl:	tvrostatus
Parameter:	
Bemerkung:	
Beschreibung:	tvrostatus => Liefert den aktuellen Status der Achsen
Rückmeldung:	„-“ = Achse „AUS“ „M“ = Achse in „Motion“ „@“ = Achse „Steht“
Fehlercode:	--
Beispiel:	tvrostatus

4.16 Konfiguration des Trigger-Ausgangssignals

Diese Befehle syncronisieren ein externes Gerät, wie z.B. eine Videokamera oder einen Laser. Diese Signale werden über den Multifunktionsport ausgegeben, welcher als Option erhältlich ist.
Wichtig!

Trigger	
Befehl:	?trig oder !trig
Parameter:	0 oder 1 (AUS / EIN)
Beschreibung:	!trig 1 ➔ Trigger „EIN“
	?trig ➔ Liefert den aktuellen Zustand der Triggerbearbeitung.
Wichtig!	Den Trigger erst einschalten, nach dem alle Einstellungen übertragen wurden. (Außer bei Triggermode 99)
Rückmeldung:	EIN oder AUS
Fehlercode:	--
Beispiel:	!trig 0 (Triggerbearbeitung „AUS“) ?trig

Trigger Achse	
Befehl:	?triga oder !triga
Parameter:	X, y, z oder a
Beschreibung:	!triga y ➔ Trigger bezogen auf die y-Achse.
	?triga ➔ Liefert die aktuelle Bezugsachse.
Rückmeldung:	X, y, z oder a
Fehlercode:	--
Beispiel:	!triga x (Trigger bezogen auf die x-Achse) ?triga

Trigger Modus			
Befehl:	?trigm oder !trigm		
Parameter:	0 – 17 oder 99		
Beschreibung:	!trigm 0 ➔		 high active
	!trigm 1 ➔		 high active
	!trigm 2 ➔		 high active
	!trigm 3 ➔		 low active
	!trigm 4 ➔		 low active
	!trigm 5 ➔		 low active
Die Trigger-Modi 6-11 erzeugen ihren ersten Trigger-Impuls nach der Hälfte der eingestellten Trigger-Strecke und dann in Abhänigkeit der Trigger-Strecke weitere Impulse.	!trigm 6 ➔		 high active
	!trigm 7 ➔		 high active
	!trigm 8 ➔		 high active
	!trigm 9 ➔		 low active
	!trigm 10 ➔		 low active
	!trigm 11 ➔		 low active
Sonderfunktion mit externem Triggersignal. (Multifunktionsport Pin 8) Das externe Triggersignal muß zum Zeitpunkt des internen Triggersignals auf „low“ liegen.	!trigm 12 ➔		 high active
	!trigm 13 ➔		 high active
	!trigm 14 ➔		 high active
	!trigm 15 ➔		 low active
	!trigm 16 ➔		 low active
	!trigm 17 ➔		 low active
Die Trigger-Modi 18 – 23 besitzen zwei Triggerausgänge. Der erste Triggerimpuls wird nach einem einstellbaren Offset (Ausgang-1: trigoffsetone) (Ausgang-2: trigoffsettow) ausgegeben. Alle weiteren Impulse werden in Abhängigkeit der eingestellten Strecke ausgegeben.	!trigm 18 ➔		 high active
	!trigm 19 ➔		 high active
	!trigm 20 ➔		high active
	!trigm 21 ➔		low active
	!trigm 22 ➔		low active

Der Offset sollte nicht größer als die Triggerdistans sein.	!trigm 23 ➔ 	 low active

!trigm 99

Bei dem Triggermode 99 wird am Anfang und am Ende der gleichförmigen Bewegung ein Triggerimpuls erzeugt. Bei der Ausführung dieser Funktion ist eine bestimmte Reihenfolge ein zu halten! Der Befehl !trigm 99 ist als letzter Befehl nach den gewohnten Triggereinstellungen zu senden, da er bei einer anderen Mode Einstellung gelöscht wird.

Rückmeldung:	0 – 23! (Mode)
Fehlercode:	--
Beispiel:	!trigm 3 (Trigger Mode 3) ?trigm

Legende				
				
Startpunkt	Triggerpunkte	Bahn	ExternesTriggersignal	 low active

Trigger Signal	
Befehl:	?trigs oder !trigs
Parameter:	0 – 5 (µs) 0 = minimaler Trigger (einige 100ns)
Beschreibung:	!trigs 4 ➔ Trigger-Signallänge 4 µs
	?trigs ➔ Liefert den aktuellen Zustand der eingestellten Trigger-Signallänge.
Rückmeldung:	0 – 5 (µs)
Fehlercode:	--
Beispiel:	!trigs 3 (Trigger-Signallänge = 3µs) ?trigs

Trigger Distanz	
Befehl:	?trigd oder !trigd
Parameter:	1 – 5000000 Motorincremente (Abhängig von der Dim)
Beschreibung:	!trigd 1 ➔ Trigger-Distance 1mm (bei Dim 2)
	?trigd ➔ Liefert die aktuelle Triggerstrecke.
Rückmeldung:	Strecke
Fehlercode:	--
Beispiel:	!trigd 3 (3mm Triggerstrecke bei Dim 2) ?trigd

Trigger Offset 1 ; Trigger Offset 2	
Befehl:	?trigoffsetone ; ?trigoffsettwo !trigoffsetone ; !trigoffsettwo
Parameter:	0 – 5000000 Motorincremente (Abhängig von der Dim)
Beschreibung:	!trigoffsetone ➔ Trigger-Distance 1mm (bei Dim 2) 20000
	?trigoffsettwo ➔ Liefert die aktuelle Triggerstrecke.
Rückmeldung:	Offset (Strecke, Grad, oder Umdrehung)
Fehlercode:	--
Beispiel:	!trigoffsettow 180 (180 Grad bei Dim 3) ?trigoffsettow

Trigger Counter; Trigger Counter 2	
Befehl:	?!trigcount; ?trigcounttwo
Parameter:	0 bis 2147483647
Bemerkung:	Es werden alle ausgegebenen Trigger gezählt
Beschreibung:	!trigcount 0 => Clear Counter 1 Ttrigcounttwo 0 => Clear Counter 2 ?trigcount => Lese Zählerstand Counter 1 ?trigcounttwo => Lese Zählerstand Counter 2
Rückmeldung:	Anzahl der ausgeführten Trigger
Fehlercode:	--
Beispiel:	?trigcount ; ?trigcounttwo (Lese Zählerstand)

4.17 Konfiguration des Snapshot-Eingangs

Mit den Befehlen können die aktuellen Positionen während des Verfahrvorgangs in der Steuerung gespeichert werden. Diese Werte können im Anschluß ausgelesen oder angefahren werden. Dieses Signal wird über den Multifunktionsport gesetzt, welcher als Option erhältlich ist.

Snapshot	
Befehl:	?sns oder !sns
Parameter:	0 oder 1
Beschreibung:	!sns 1 ➔ Snapshot „EIN„
	?sns ➔ Liefert die aktuelle Snapshot-Zustand.
Rückmeldung:	Snapshot-Zustand
Fehlercode:	--
Beispiel:	!sns 0 (Snapshot „AUS„) ?sns

Snapshot-Level (Polarität)	
Befehl:	?snsl oder !snsl
Parameter:	0 oder 1
Beschreibung:	!snsl 1 ➔ Snapshot ist high-aktiv. 
	?snsl ➔ Liefert die aktuelle Polarität
Rückmeldung:	Aktuelle Polarität
Fehlercode:	--
Beispiel:	!snsl 0 (Snapshot ist low-aktiv)  ?snsl

Snapshot Filter	
Befehl:	?snsf oder !snsf
Parameter:	0 – 100 ms
Bemerkung:	Dient als Eingangsfilter bei prellenden Schaltern
Beschreibung:	!snsf 10 => 10 ms Eingangsfilter ?snsf => Liefert den aktuellen Wert
Rückmeldung:	Aktuelle Filterzeit
Fehlercode:	--
Beispiel:	!snsf 0 (Kein Eingangsfilter) ?snsf

Snapshot-Modus	
Befehl:	?snsm oder !snsm
Parameter:	0 oder 1
Beschreibung:	!snsm 1 ➔ Snapshot „Automatik“ Die Position wird nach dem ersten Impuls automatisch angefahren.
	?snsm ➔ Liefert den aktuellen Mode
Rückmeldung:	Snapshot-Mode
Fehlercode:	--
Beispiel:	!snsm 0 (Normaler Snapshot) ?snsm

Snapshot-Zähler	
Befehl:	?snsc
Parameter:	–
Beschreibung:	Inhalt wird nach jedem „lesen„ gelöscht.
	?snsc ➔ Liefert die Anzahl der ausgelösten SnapShot`s.
Rückmeldung:	Liefert die Anzahl der ausgelösten SnapShot`s
Fehlercode:	--
Beispiel:	?snsc

Snapshot-Position	
Befehl:	!snsp oder ?snsp
Parameter:	X,y,z und a Min-/max-Verfahrbereich
Bemerkung:	Ein- und Ausgabe ist abhängig von der Dimension.
Beschreibung:	!snsp 1000 2000 3000 ➔ Positionswerte für die Achsen x, y und z werden gesetzt.
	!snsp y 2000 ➔ Position der y-Achse wird gesetzt.
	?snsp ➔ Abfrage der aktuellen Snapshot-Position aller Achsen.
	?snsp z ➔ Abfrage der aktuellen Snapshot-Position von Achse z.
Rückmeldung:	Positionswerte
Fehlercode:	--
Beispiel:	!snsp 100 200 (Setzen der Positionen von x- und y-Achse) ?snsp (Abfrage der Snapshot-Positionen aller Achsen)

Snapshot-Position-Array	
Befehl:	?snsa
Parameter:	X,y,z und a 1 – 200 (Positionen)
Bemerkung:	Ein- und Ausgabe ist abhängig von der Dimension.
Beschreibung:	?snsa 33 ➔ Abfrage der Snapshot-Position 33 aller Achsen.
	?snsa z 99 ➔ Abfrage der Snapshot-Position 99 von Achse z.
Rückmeldung:	Positionswerte
Fehlercode:	--
Beispiel:	?snsa 1 (Abfrage der Snapshot-Positionen 1 aller Achsen)

Snapshot-Offset	
Befehl:	!?snso
Parameter:	x,y,z and a
Bemerkung:	Nur für Automatikbetrieb
Beschreibung:	?snso!snso 2 0 0
Rückmeldung:	Eingestellter Wert
Fehlercode:	--
Beispiel:	!snso –2 0 1 (die X-Achse wird um 2mm zurück gefahren und die Z-Achse fährt um 1mm vor, wie die gespeicherte Position.

5 Anhang Allgemein

5.1 Die Pinbelegung des Multifunktionsport *(Nicht bei ECO-STEP)*

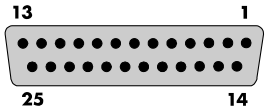


Abb.: Der Multifunktionsport (25 Pol Sub-D Buchse)

Bedingt durch die Funktionsvielfalt sind die Pins des Multifunktionsport *(MFP)* teilweise mehrfach belegt. Je nach Ausstattung der Steuerung bedeutet dies, dass jeweils nur ein Signalaus- bzw. eingang auf einem Pin des MFP anliegt.
Die gewünschte Funktionalität muß bei der Bestellung mit angegeben werden.
Standart ist: Trigger, Snapshot, und Stopeingang

Pin	Signal		Bemerkungen
1	Takteingang X	Standard:	TTL-Pegel
	Taktausgang X	Sonderfunktion:	TTL-Pegel
	Gebereingang X Spur A	Sonderfunktion:	+5V U-low ≤ 0,8V / U-High ≥ 3,6V
2	Vor-/Rück X Eingang	Standard:	TTL-Pegel
	Vor-/Rück X Ausgang	Sonderfunktion:	TTL-Pegel
	Gebereingang X Spur B	Sonderfunktion:	+5V U-low ≤ 0,8V / U-High ≥ 3,6V
3	Takteingang Y	Standard:	TTL-Pegel
	Taktausgang Y	Sonderfunktion:	TTL-Pegel
	Gebereingang Y Spur A	Sonderfunktion:	+5V U-low ≤ 0,8V / U-High ≥ 3,6V
4	Vor-/Rück Y Eingang	Standard:	TTL-Pegel
	Vor-/Rück Y Ausgang	Sonderfunktion:	TTL-Pegel
	Gebereingang Y Spur B	Sonderfunktion:	+5V U-low ≤ 0,8V / U-High ≥ 3,6V
5	Takteingang Z	Standard:	TTL-Pegel
	Taktausgang Z	Sonderfunktion:	TTL-Pegel
	Gebereingang Z Spur A	Sonderfunktion:	+5V U-low ≤ 0,8V / U-High ≥ 3,6V
	Tigger out 2	Standard:	TTL-Pegel / I_{max} = 1,6 mA
6	Analoger Eingang Ain 10 Channel 10	Sonderfunktion:	Messbereich 0...0,5V/ Ri = 1,1 kΩ

Pin	Signal		Bemerkungen
7	Start / Stopp Z	Standard:	TTL-Pegel Start | Stopp Freigabe für Takt Vor- / Rück
	Ain 8 / Channel 8	Sonderfunktion:	0...5V
8	Start / Stopp X	Standard:	TTL-Pegel Start | Stopp Freigabe für Takt Vor- / Rück
	Ain 6 / Channel 6	Sonderfunktion:	0...5V
9	- 12V		I_{max} = 20mA
10	Joystick ein	Standard:	Externer Umschalter „MAN/AUTO“
11	VAGND	Standard:	Masse der +5V Referenzspannung
12	Joystick Y Ain 1 / Channel 1	Standard:	liegt parallel zu ST1 Pin 4
13	VAREF	Standard:	+5V Referenzspannung
14	Vor-/Rück Z Eingang	Standard:	TTL-Pegel
	Vor-/Rück Z Ausgang	Sonderfunktion:	TTL-Pegel
	Gebereingang Z Spur B	Sonderfunktion:	+5V U-low ≤ 0,8V / U-High ≥ 3,6V
15	Tigger out	Standard:	TTL-Pegel / I_{max} = 1,6 mA
16	GND		
17	+5V		I_{max} = 300 mA
18	Analoger Ausgang Channel 2	Standard:	Analogausgang 0...10V bzw. +/-10V je nach Bestückung, Ri_{min} = 1kOhm / I_{max} =10mA
	Digitaler Ausgang	Sonderfunktion:	TTL-Pegel
19	Ain 9 / Channel 9	Sonderfunktion:	Messbereich 0...0,5V/ Ri = 1,1 kΩ
20	Start / Stopp Y	Standard:	TTL-Pegel Start | Stopp Freigabe für Takt Vor- / Rück
	Ain 7 / Channel 7	Sonderfunktion:	0...5V
21	+12V		I_{max} = 500 mA
22	SnapShot Eingang	Standard:	TTL, Pull Up = 4,7 kOhm, RC-Filter 470 Ohm/100nF
23	Stopp Eingang	Standard:	TTL, Pull Up = 4,7 kOhm, RC-Filter 470 Ohm/100nF
24	Joystick X Ain 0 / Channel 0		liegt parallel zu ST1 Pin 3
25	Joystick Z Ain 2 / Channel 2		liegt parallel zu ST1 Pin 5

5.2 Die Pinbelegung der RS232 Schnittstelle

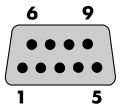


Abb.: Die RS232 Schnittstelle (9 Pol Sub-D Buchse)

Pin	Signal	Bemerkungen
1	n.c.	
2	RxD	Empfängerleitung LSTEP
3	TxD	Sendeleitung LSTEP
4	GND	
5	GND	Signalmasse
6	+5V	
7	RTS	Request to send, von LSTEP
8	CTS	Clear to send, von PC
9	entweder n.c. +5V od. +12V DC	

5.3 Das Schnittstellenkabel

LSTEP		PC		
9 Pol Sub-D Stecker	Belegung	9 Pol Sub-D	25 pol Sub-D	Belegung
1	n.c.	-	-	-
2	RxD	3	2	TxD
3	TxD	2	3	RxD
4	n.c.	-	-	-
5	GND	5	7	GND
6	n.c.	-	-	-
7	RTS	8	5	CTS
8	CTS	7	4	RTS
9	n.c.	-	-	-

5.4 Die Pinbelegung des Joystick Anschluß

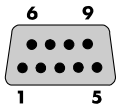


Abb.: Der Joystick Anschluß (9 Pol Sub-D Buchse)

Pin	Signal	Bemerkungen
1	GND	
2	Joystick Ein	
3	X-Achse	Schleifkontakt des Joystick
4	Y-Achse	Schleifkontakt des Joystick
5	Z-Achse	Schleifkontakt des Joystick
6	n.c.	
7	n.c.	
8	VAref (+5V)	5V Analoge Referenzspannung
9	VAref (+5V)	5V Analoge Referenzspannung

5.5 Die CAN Schnittstelle (Nicht bei ECO-STEP)

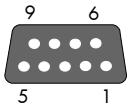


Abb.: Die CAN-Schnittstelle (9 Pol Sub-D Stecker)

Um mehr als eine Steuerung vom Typ LSTEP-xx/2 an einem einzigen PC betreiben zu können, wird die CAN Schnittstelle eingesetzt. Hierbei handelt es sich um eine sehr schnelle serielle Verbindung mit Datenraten bis zu **5MBd**. Um den PC mit einer solchen Schnittstelle auszurüsten, ist in der Regel eine zusätzliche Einsteckkarte erforderlich.
Theoretisch ist es möglich, bis zu **254** verschiedene Steuerungen LSTEP-xx/2 oder auch andere Geräte, die einen CAN Anschluß besitzen, miteinander zu vernetzen.
Physikalisch wird diese Schnittstelle als verdrillte Zweidrahtleitung nach RS 485 ausgeführt.
Achtung! Zur Zeit wird noch kein CAN-Protokoll unterstützt.

Pin-Nr.	Belegung	Pin-Nr.	Belegung
1	n.c.	6	CAN GND
2	CAN L	7	CAN H
3	CAN GND	8	n.c.
4	n.c.	9	CAN V+ (J2 gesteckt: +12V)
5	CAN Schirm (GND)	10	n.c.

5.6 Der Handrad-Anschuß (Koaxtrieb)

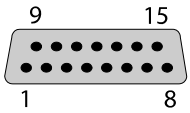


Abb.: Der Handrad-Anschlus (15 Pol Sub-D Buchse)

Pin	Belegung	Pin	Belegung
1	Analog VCC (+5V)	9	Analog GND
2	+5V	10	Analog GND
3	A+, X-Achse	11	C+, Y-Achse
4	A-, X-Achse	12	C-, Y-Achse
5	B+, X-Achse	13	D+, Y-Achse
6	B-, X-Achse	14	D-, Y-Achse
7	TTL-Eingang Auflösung	15	n.c.
8	TTL-EingangSnap-Shot	Gehäuse	Schirm

5.7 Interpreter für MULTICONTROL-Kommandos

Optional kann die LSTEP-xx/2 auch Multicontrol-Kommandos verarbeiten.
Um auf diesen Befehlssatz umzuschalten, kann entweder der Schalter Nr. 2 des Dip-Switch auf „ON“ geschaltet werden, oder er wird per Befehl umgeschaltet
(siehe Kap.4 „Interpreter“).
Um den Befehlssatz per Dip-Switch umzuschalten ist die Steuerung zuvor Spannungsfrei zu schalten. Der Dip-Switch befindet sich bei beiden Steuerungen in der Rückplatte.

Abb.: Der Dip-Switch der LSTEP-xx/2

Abb.: Der Dip-Switch der ECO-STEP

5.7.1 Eingabe von Parametern

Parameter können als Integer- oder Gleitkommawerte eingegeben werden.
Gleitkommawerte werden nur im Dezimalformat unterstützt. Das wissenschaftliche Format kann **nicht** verwendet werden.

23.45676	Eingabe wird unterstützt
34.e01	Eingabe wird nicht unterstützt
0.67E-1	Eingabe wird nicht unterstützt

5.7.2 Unterstützte Multicontrol-Kommandos

Zur Zeit werden die folgenden Venus-Kommandos unterstützt:

setdim	setlimit	setaccel	calibrate	setsw
getdim	getlimit	getaccel	rmeasure	getsw
geterror	setpitch	setaxis	move	setcalvel
setvel	getpitch	getaxis	rmove	getcalvel
getvel	setpos	version	pos	setrmvel
joyspeed	setjoysticktype	identify	devpos	getrmvel
joystick	getjoysticktype	status	getpos	

Wird ein Venus-Befehl übertragen den die Steuerung nicht interpretieren kann, wird der Fehlercode auf 9999 gesetzt. Weitere Aktionen werden nicht ausgeführt.

Folgende Befehle werden z.Zt nicht unterstützt:

align	getunit	setunit	Ico	scale
getec	setcloop	getcloop	Setclfactor	getclfactor
echo	Sowie alle Befehle die den Stack und die Kette benutzen.			

Abweichungen und Unterschiede

Einige Befehle des LSTEP Venus-Interpreters arbeiten etwas anders als bei einer MultiControl. Dies betrifft vor allem Kommandos, die den internen Staus anzeigen bzw. die Versionsnummer von Steuerung oder Firmware zurückliefern. Im folgenden sind die bekannten Abweichungen dokumentiert:

Venus-Kommando	Bedeutung bei MultiControl	Bedeutung bei LSTEP-xx/2
version	liefert als Rückmeldung die Versionsnummer des VenusInterpreters	liefert als Rückmeldung die Versions- und Revisionsnummer des ITK-Interpreters.
identify	liefert als Rückmeldung die Identifikation des Controllers, die Schalterstellung an der Rückwand während des Einschaltens, die internen Konfigurationsschalterstellungen, die Hardware- und die SoftwareRevisionsnummer	liefert als Rückmeldung die Versions- und Revisionsnummer des ITK-Interpreters. Der zurückgesendete String enthält in den ersten 4 Zeichen kodiert die Versionsnummer des ITK-Interpreters. Die nächsten beiden numerischen Ziffern geben die Revisionsnummer der Version an. Bsp.: *1.00-12 99 99 3d* bedeutet Vers. 1.00, Rev. 12
setdim	Setzt die Dimension der Position für die Befehle mit den Parametern in []	wie bei MultiControl. Wird bei Befehlen die falsche Anzahl von Parametern übergeben, so werden diese Befehle nicht ausgeführt
status	liefert als Rückmeldung den momentanen Status der MultiControl	liefert als Rückmeldung immer 0 *Hinweis:* stattdessen kann das Status-Register der LSTEP ausgelesen werden
mode	Setzt den interaktiven Modus des Venus-Interpreters 1 = Terminal Mode 0 = Host Mode	Die LSTEP-xx arbeitet ausschließlich im Host-Mode. Daher führt das Kommando *1 mode* zum Fehlercode 1003
save	speichert alle Parameter mit der Kennzeichnung nv im nichtflüchtigen Speicher	ein Speichern der eingestellten Parameter im nichtflüchtigen Speicher wird z. Zt. nicht unterstützt. Wird dieser Befehl aufgerufen, so wird der Fehlercode 1200 *'Schreibfehler im Flashspeicher'* gesetzt
restore	überschreibt alle Parameter mit der Kennzeichnung nv mit den im nichtflüchtigen Speicher abgelegten Werten	ein Auslesen der eingestellten Parameter im nichtflüchtigen Speicher wird z. Zt. nicht unterstützt. Wird dieser Befehl aufgerufen, so wird der Fehlercode 1202 *'Lesefehler im Flashspeicher'* gesetzt
setunit	setzt die Einheit einer Achse in physikalischen Größen	0 = Motorinkremente 1 = µm 2 = mm

Venus-Kommando	Bedeutung bei MultiControl	Bedeutung bei LSTEP-xx/2
setpitch	r, i, setpitch: Setzt den Wert für die Spindelsteigung der Achse i auf r	r, i, setpitch: Setzt den Wert für die Spindelsteigung der Achse i auf r. Die Geschwindigkeitsachse (i=0) wird nicht unterstützt.
move	absolut Positionieren. Bei Überschreiten des zulässigen Verfahrbereiches wird der Fehlercode 1004 gesetzt.	absolut Positionieren. Die Überschreitung des Verfahrbereichs wird überwacht und ggfs. auf den maximal zulässigen Verfahrbereich begrenzt. Die LSTEP setzt den Fehlercode nicht auf 1004.
selftest	liefert das Ergebnis nach dem Selbsttest für eine Achse oder den Controller	liefert immer 0 zurück
setjoysticktype	definiert den angeschlossenen Joysticktyp	der Joysticktyp richtet sich immer nach der der Achsanzahl angeschlossenen Steuerung. Bei einer 2-Achsensteuerung wird immer von einem Zwei-Achsenjoystick, bei einer 3 Achsen Steuerung immer von einem Drei-Achsenjoystick ausgegangen. Aus Kompatibilitätsgründen mit Anwendungsprogrammen für Steuerungen vom Typ multicontrol wird der Befehl hier zugelassen. Die LSTEP führt diesen Befehl ohne Fehlermeldungy aus. Er hat jedoch keine Wirkung, ausser daß der hierüber eingestellte Wert mit *getjoysticktype* zurückgelesen werden kann.
getjoysticktype	liefert den aktuellen Joysticktyp zurück	liefert den zuletzt mit *setjoysticktype* gesetzten Wert zurück
setjoyspeed **joyspeed**	setzt die Geschwindigkeit für den Joystick	hat die gleiche Wirkung wie der Befehl *joyspeed*. Die maximale Joystickgeschwindigkeit wird in Motor-Umdrehungen/sec angegeben. Bsp.: 13 setjoyspeed setzt die maximale Geschwindigkeit auf 13 Umdrehungen /sec
getjoyspeed	liefert die aktuelle Joystickgeschwindigkeit zurück	liefert die maximale Geschwindigkeit bei voller Auslenkung des Joysticks zurück. Es ist diejenige Geschwindigkeit, die mit den Befehlen *joyspeed* bzw. *setjoyspeed* eingestellt worden ist

Venus-Kommando	Bedeutung bei MultiControl	Bedeutung bei LSTEP-xx/2
setmotortype **getmotortype**	ordnet den Achsen bestimmte Motortypen zu nur für Service Zwecke	Die LSteps sind werksseitig so eingestellt, daß sie für alle gängigen Motortypen ohne Anpassung verwendet werden können. Daher sind die Befehle *setmotortype* sowie *getmotortype* hier nicht notwendig und werden daher nicht unterstützt..
setcurrent, **getcurrent**	nur für ServiceZwecke	Setzt oder liest den Ausgangsstrom der Achsen.
v t setcalvel	definiert die Kalibriergeschwindigkeit v in Umdrehungen/sec beim Anfahren der Endschalterposition (t=1) und Zurückfahren aus der Endschalter-position (t=2)	definiert die Kalibriergeschwindigkeit v in Umdrehungen/sec beim Anfahren der Endschalterposition (t=1). Die Einstellung der Geschwindigkeit beim Zurückfahren aus der Endschalter-position (t=2) ist bei den LSteps fest eingestellt. Daher wird der Befehl mit dem Parameter t=2 nicht unterstützt. Es wird die Fehlernummer 9999 gesetzt.
[r] setpos	setzt die aktuelle Position auf [r]	setzt die aktuelle Position auf [r]. Dies entspricht einer Verschiebung des benutzten Koordinatensystems.
getpos	liefert die Nullpunktsverschiebung in Microsteps zurück	liefert die Verschiebung des mit *setpos* benutzerdefinierten Koordinatensystems bezogen auf den Nullpunkt nach dem Kalibrieren in Microsteps zurück
pos	liefert die aktuelle Position in dem aktuellen Koordinatensystem zurück	liefert die Position innerhalb des aktuellen (mit *setpos* verschobenen) Koordinatensystems in mm zurück.
devpos	liefert die aktuelle Position in Micro-steps vom Nullpunkt zurück	liefert die Position innerhalb des mit *setpos* verschobenen Benutzer-Koordinatensystems in Microsteps zurück.
m n l setsw	m = 0 definiert Endschalter n als Schließer m = 1 definiert Endschalter n als Öffner n = 0 bedeutet Kalibrierendschalter n = 1 bedeutet Endendschalter l = 1,2,3 bedeutet Achsnummer	Bei der LSTEP sind alle Endschalter-eingänge so beschaltet, daß in Abhängigkeit des eingesetzten End-schaltertyps folgende Zuordnung gilt: Öffner: beim Überfahren des Endschalters erscheint eine Flanke von 0 nach 1 Schließer: beim Überfahren des Endschalters erscheint eine Flanke von 1 nach 0. Es gilt dieselbe Zuordnung für m, n und l wie bei der multicontrol

5.8 Der Motoranschluß

Die LSTEP-xx/2 ist vorwiegend für den Betrieb mit leichten Koordinatentischen, angetrieben durch 2-Phasen-Schrittmotoren bis 5 A, konzipiert. Mit der hochauflösenden Ansteuerung und Beschleunigung über Rampen in allen Betriebsarten (auch Joy-Stick) ist ein sanfter schonender Ablauf gewährleistet. Für den sicheren Betrieb sollten Sie jedoch noch folgende Punkte beachten:

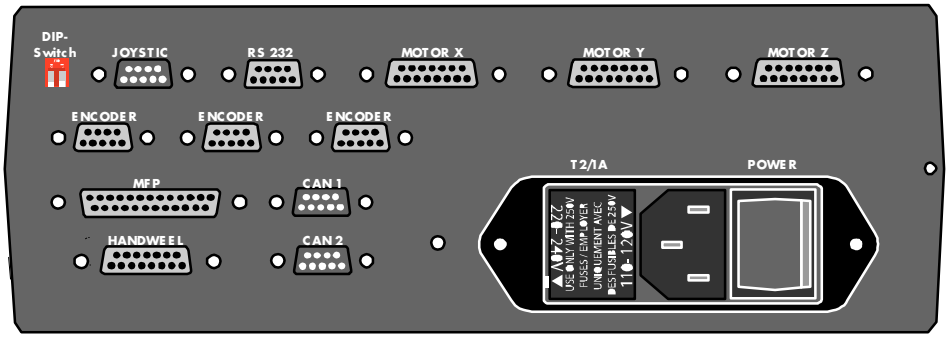
- Motoren niederohmig mit geringen Induktivitäten wählen.
- 8-Leiter-Motor niederohmig schalten.
- der Motorstrom sollte jedoch zur Vermeidung unnötiger Wärmefehler immer nur so hoch wie nötig eingestellt werden.
- Motorstrom in Höhe des Nennstromes führt zur Sättigung des Magnetmaterials und die Schrittwinkelfehler nehmen zu.

5.9 Fehlersuchanleitung

Fehlerbeschreibung	Fehlereingrenzung / Fehlerbeseitigung
1 Totalausfall	Netzanschluss und Netzsicherung in Eurobuchse an Geräterückseite prüfen
2 Motor wird zu warm	Verschaltung des Motors prüfen (vgl. Motoranschluss)
3 Motor läuft nicht mit hohen Drehzahlen	Motor ist zu hochohmig (vgl. Motoranschluss)
4 Einzelner Motor brummt und steht trotz niedrig eingestellter Drehzahl	Motorkabel am Tisch gegeneinander tauschen, bleibt der Fehler in der gleichen Achse: - Kabel und Motor prüfen; ist der Fehler in der anderen Achse: - Fehler in der LSTEP
5 Einzelne Achse läuft nicht, keine Brummgeräusche	a) Endschalter prüfen b) prüfen gem. 4
6 Keine Datenverbindung über RS 232	a) Spannungen an der LSTEP bei abgezogenem Interface-Kabel prüfen b) Rechner und Interface-Kabel prüfen
7 Rückmeldungen der LSTEP sind versetzt. Erst nach mehrmaligem Lesen kommt die richtige Meldung	Meldung der LSTEP wurde nicht aus Empfangspuffer gelesen, Anwenderprogramm prüfen; nach einem Start- oder Lesebefehl wurde die Antwort der LSTEP ignoriert

6 Anhang LSTEP

6.1 Die Rückwand der LSTEP



6.2 Motoranschluß X/Y/Z

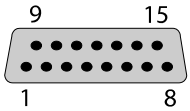


Abb.: Der Motoranschlus (15 Pol Sub-D Buchse)

15 pol. D-SUB an LSTEP,Pin Nr.	Farbe	12-pol. Flanschdose, Motor	Pin Belegung:
1 + 9	blau	K	Phase 1R
2 + 10	orange	J	Phase 1T
3 + 11	weiß	B	Phase 2T
4 + 12	braun	C	Phase 2R
5	gelb	G	Endschalter Endposition
6	grau	H	Endschalter Nullposition
7	rot	A	+5V
8	schwarz	F	GND
13	grün	E	Referenzschalter
14 **(bei der X- u. Y-Achse!)**	violett	D	Temperatur
14 **(bei der Z-Achse!)**	violett	D	Optionale Spannung für eine Motorbremse.
15			+12V

6.3 Der Geberanschluß X/Y/Z *(Nicht bei ECO - STEP)*

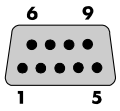


Abb.: Der Geberanschluß (9 Pol Sub-D Buchse)

PIN	Signal	PIN	Signal
1	U_{1-}	6	U_{1+}
2	0V	7	5V
3	U_{2-}	8	U_{2+}
4	+12V (Optional)	9	U_{0+}
5	U_{0-}	Gehäuse	Außenschirm

6.4 Das Netzanschlußmodul

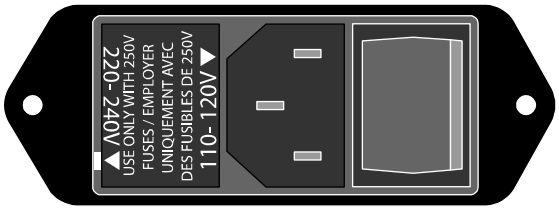


Abb.: Das Netzanschlußmodul

Der Netzanschluß beinhaltet den Kaltgerätestecker, den Netzschlater und den Spannungswähler mit integrierten Netzeingangs-Sicherungen.
Die Steuerung kann alternativ mit 220V-240V oder 110V-120V betrieben werden. Hierzu ist es zwingend erforderlich, den Spannungswähler entsprechend zu stecken. Der Pfeil der gewünschten Spannung muß auf die weiße Markierung zeigen.
Um eine Sicherung zu wechseln ist der Spannungswähler aus dem Netzanschlußmodul herauszuziehen. Bei einer Spannung von 220V-240V ist eine träge Sicherung von 1Ampere einzusetzen, während bei einer Spannung von 110V-120V eine träge Sicherung von 2Ampere zu verwenden ist. (Dies gilt für die LSTEP-1x/2 + 2x/2) Es gilt in beiden Fällen die Seite des Pfeiles der entsprechenden Spannung.

6.5 Belegung der DIP-Switches

Abb.: Die DIP-Switches der LSTEP

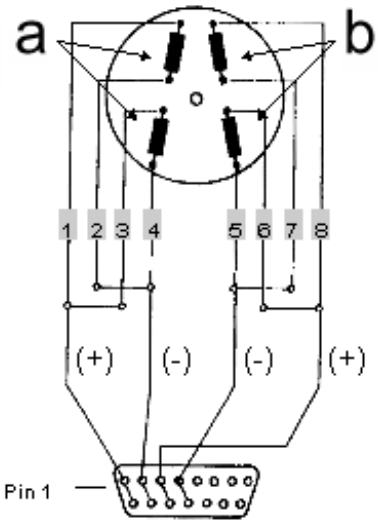
Schalter 1 ON ➔ Firmware update eingeschaltet
OFF ➔ Firmware update ausgeschaltet
Schalter 2 ON ➔ Multicontrol-Befehlssatz
OFF ➔ Standard-Befehlssatz

6.6 Technische Daten

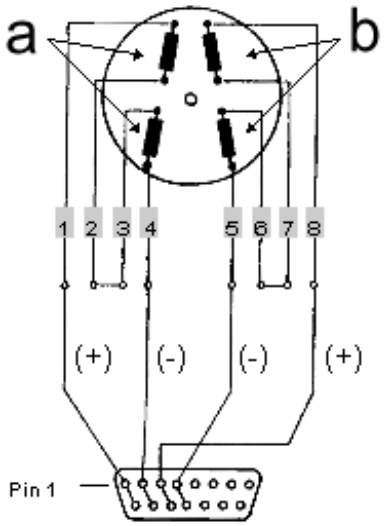
Netzanschluss:	110V - 120V / 200V - 240V +/-10% 50/60Hz, 100VA
Sicherungen:	
- primär (in Eurobuchse):	• 2 A träge / 1 A träge LSTEP-1 und LSTEP-2 • 5 A träge / 2,5 A träge LSTEP-3
- sekundär (auf der Platine)	Si1 LSTEP-1 und 2 ➔ 5 A träge / LSTEP-3 ➔10A träge
Max. Netzausfalldauer:	< 50ms bei Netzausfall (<0,77 * U_N) schaltet die LSTEP auf Reset
Max. Motordrehzahl:	40 U/sec. bei 200-schrittigem Motor
Max. Motorstrom:	1,25A je Motorphase für LSTEP-1 2,5A je Motorphase für LSTEP-2 5,0A je Motorphase für LSTEP-3
Max. Motorspannung:	40V
Schrittauflösung	• max. 50.000 (100.000) Schritte/Umdrehung bei 200schrittigem Motor. • 2000 Schritte/Vollschritt bei Linearschrittmotoren
Baudrate:	9600, 19200, 38400, 57600 oder 115200
Umgebungsbedingungen:	
Lufttemperatur bei Betrieb:	15 ... 40 Grad C
Lufttemperatur ausser Betrieb:	0 ... 43 Grad C
Luftfeuchtigkeit bei Betrieb	8 ... 80 % bei 31° / Maximal 50% bei 40°
Luftfeuchtigkeit ausser Betrieb	0 ... 80 %
Abmessungen B * T * H (ohne Tragegriff):	
	250 mm • 230 mm • 100 mm bei LSTEP-1x
	250 mm • 230 mm • 100 mm bei LSTEP-2x
	475 mm (19“) • 266 mm • 90 mm (2HE) bei LSTEP-3x
Gewicht:	4,5 kg / LSTEP-1 und LSTEP-2
	9 kg / LSTEP-3

6.7 Beschaltung des Motors

2-Phasenmotor, niederohmig



2-Phasenmotor, hochohmig



15 pol. Stecker oder Flanschstecker

6.8 Prüf- und Abgleichanleitung

Nach der folgenden Anleitung kann die LSTEP xx/2 geprüft und eingestellt werden. Die Arbeiten sind für ausgebildetes Fachpersonal vorgesehen.

ACHTUNG: **Vor Öffnen des Gerätes Netzstecker ziehen!**

Jumper 5:	Referenzspannung (+5V +/-5%)
Lötbrücke 11	geregelte Logikspannung (+4,8V...5,25V)
Lötbrücke 10	geregelte Logikspannung (-12V +/-5%)
Lötbrücke 8	geregelte Logikspannung (+12V +/-5%)
Messpunkt 15	Motorspannung 40 Volt

Prüfen des Motorstroms mit dem Oszilloskop (X/Y-Darstellung):

- X-Motorstrom:
  Oszilloskop an Messpunkt 5 und Messpunkt 6 anschließen.
- Y-Motorstrom:
  Oszilloskop an Messpunkt 9 und Messpunkt 10 anschließen.
- Z-Motorstrom:
  Oszilloskop an Messpunkt 12 und Messpunkt 13 anschließen.

Hinweis:	Die gemessene Spannung Us (Kreisradius) und **nicht** Uss (Kreisdurchmesser) entspricht dem Motorstrom

LSTEP-1x /2	6V/A, max. 1,25A
LSTEP-2x /2	3V/A, max. 2,5A
LSTEP-3x /2	1,5V/A, max. 5,0A

Joy-Stick - Abgleich
Der Joystick-Abgleich erfolgt automatisch durch die Steuerung.

Hinweis:	Beim Einschalten der Steuerung darf der Joystick nicht ausgelenkt werden, da sich die Steuerung auf die Nullstellung kalibriert.

6.9 Die Platinenansicht

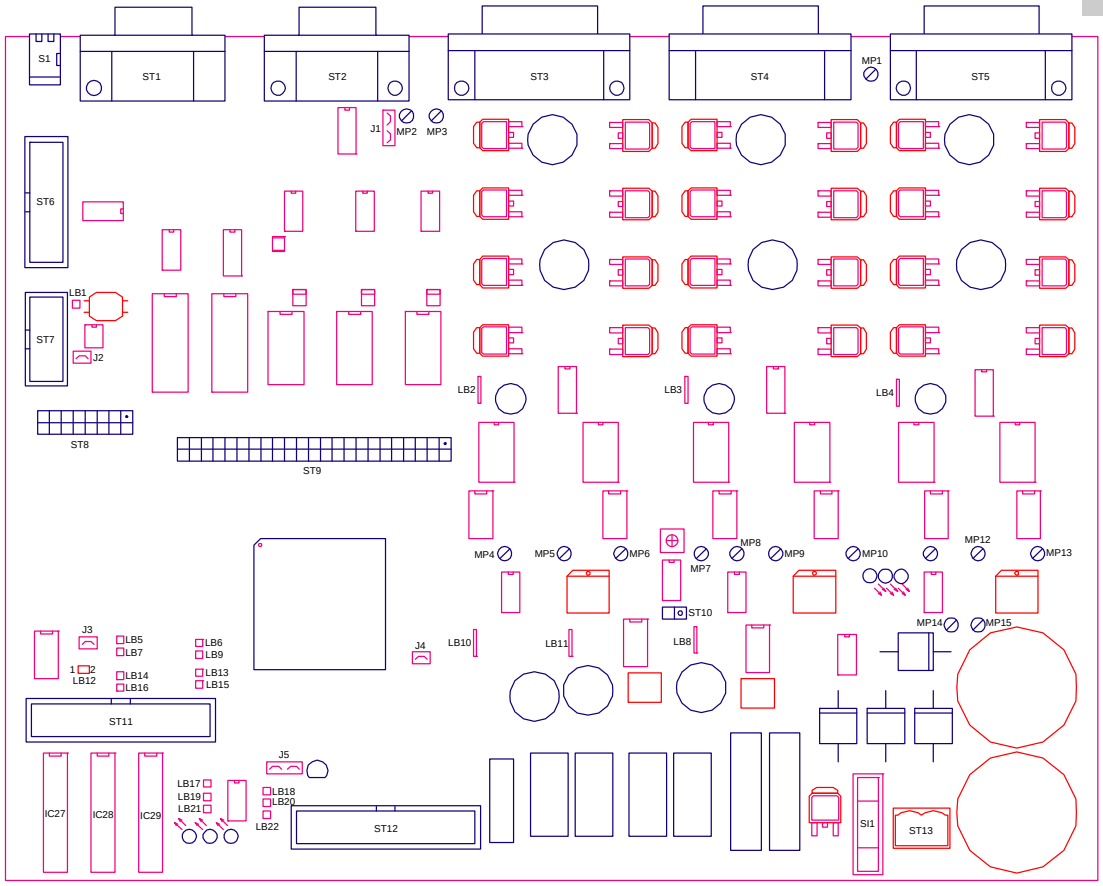


Abb.: Die Hauptplatine

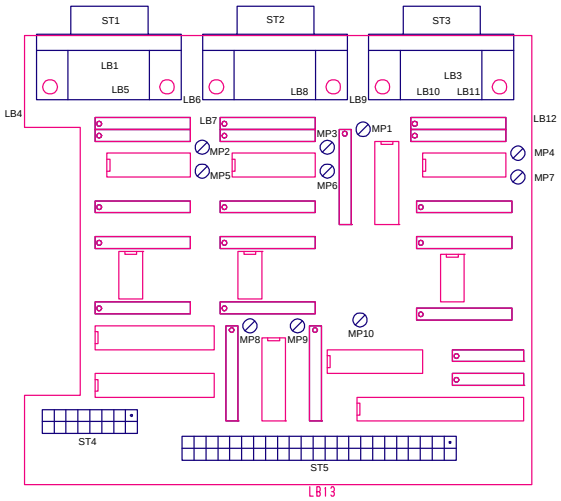
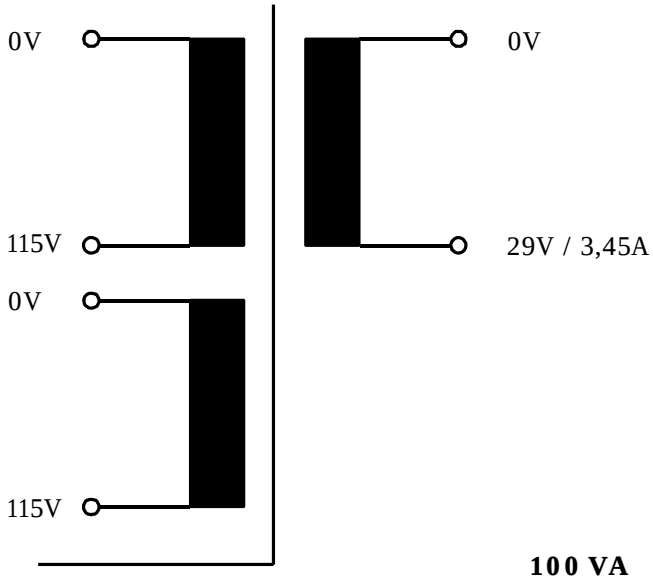


Abb.: Die Geberplatine (Optional)

Hinweis:	Die Lötbrücken der Geberkarte befinden sich auf der Lötseite der Platine.

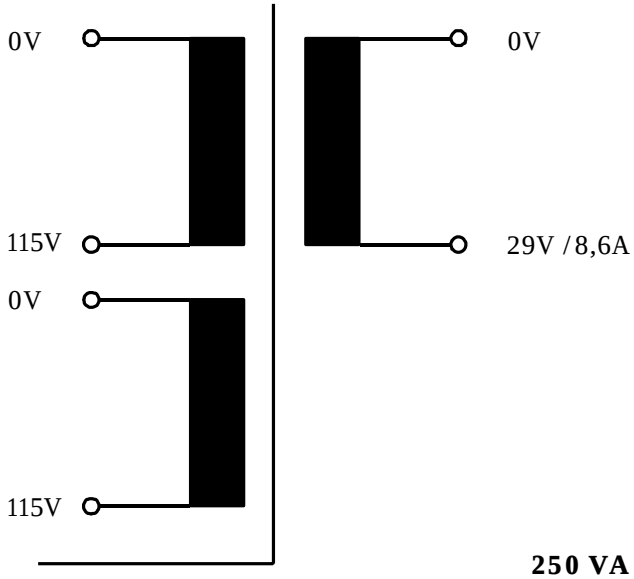
6.10 Transformatorbeschaltung

LSTEP-1x/2 ; LSTEP-2x/2



100 VA

LSTEP-3x/2



250 VA

6.11 I/O - Karte für LSTEP Steuerungen

Beschreibung 16 Ein-, 16 Ausgänge und 2 analoge Ausgänge für LSTEP 46-pin Busadapter

Die 16 Ein- und 16 Ausgänge und 2 analoge Ausgänge umfassende Karte ist für LSTEP 46-pin Pfostenleisten-Busadapter geeignet.
Die Belegung von ST2 weicht von der Belegung des I/O-Steckers der LSTEP-PC-Karte ab. Grund: Bei einer Flachbandleitung können die einzelnen Adern jeweils nur mit einem Strom von 1A beaufschlagt werden. Bei LSTEP-PC wird die Versorungsspannung von außen zugeführt. D.h. die +11,4...32V-Leitung trägt den gesamten Strom, während die GND-Leitung nahezu unbelastet ist. Die +11,4...32V-Leitung ist deshalb 4-fach ausgeführt. Bei der vorliegenden Karte wird der Strom auf der Karte eingespeist (ST4). Dadurch wird die +11,4...32-V-Leitung nahezu nicht belastet, während die GND-Leitung als Rückleiter den gesamten Strom trägt. Sie wird deshalb hier 4-fach ausgeführt. Bei externer Stromversorgung muß die 11,4...32V-Versorgungsspannung unbedingt über ST4 eingespeist werden. Eine Einspeisung über ST2 ist unzulässig.

ST1: Belegung des 46-pin-Busadapters:

Pin Nr.	Funktion
1-9	D0 – D9
18 - 20	A1 - A3
23 - 34	A6 - A17
35	/RD
36	/WR
37	- 12V
38	+ 12V
39	+ 5V
40	GND
42	/RSTOUT

ST3: 10-pol Pfostenleiste mit D-Sub-Buchse-Belegung: 2 analoge Ausgänge

Die Ausgänge sind standardmäßig für +/- 10V ausgelegt. Andere Ausgangsspannungsbereiche (z.B. +/- 5V, 0...5V, 0...10V,...) sind auf Anfrage möglich. Die Ausgänge sind mit +/- 5mA belastbar. Der Innenwiderstand beträgt ca. 100 Ohm.

Pin Nr.	Funktion
1,2	GND
3	Ausgang 1
4	Ausgang 2

Bestückung der Platine bei verschiedenen Spannungsbereichen der Analogausgänge

Ausgangsspannung	R1/R2	R4/R5	R6/R7
0...5V	10kOhm		10kOhm
0...10V	10kOhm		20kOhm
-5V...+5V	10kOhm	20kOhm	20kOhm
-10V...+10V	10kOhm	20kOhm	39 (40)kOhm

ST4: 2-pol Powerstecker für die Versorgung der Ein- und Ausgänge

Die Powerversorgung wird auf der Platine per Feinsicherung 5x20mm abgesichert. Der Auslösestrom der Sicherung darf 4A flink nicht überschreiten.

Pin Nr.	Funktion
1	+11,4...32V
2	0V

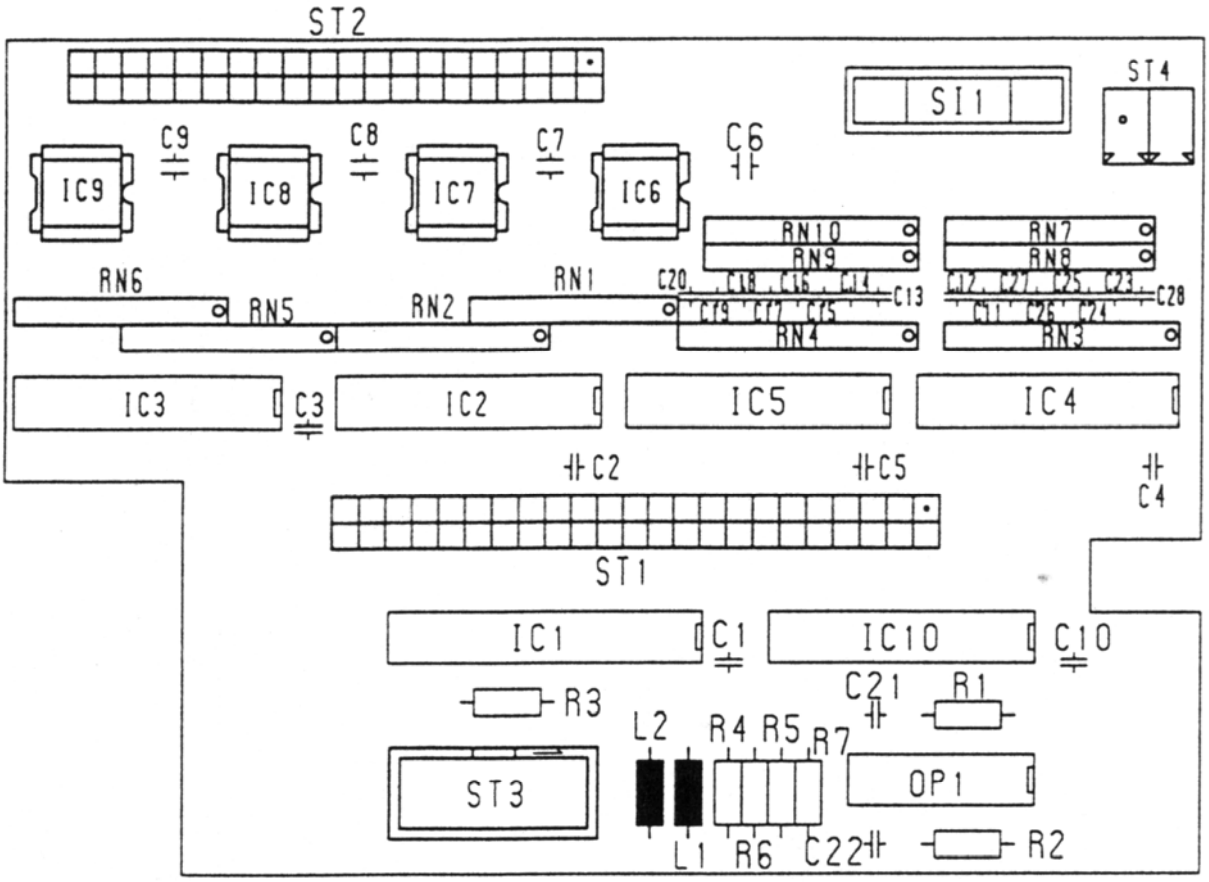
ST2: 40-pol Pfostenleiste mit 37-pol D-Sub-Stecker-Belegung: 16 Eingänge, 16 Ausgänge

Eingänge: 0...3V = „L“, 10...32V = „H“, Ri = ca. 3,3kOhm
Ausgänge: Schaltend nach +Ub=11,4...32V, Imax = 0,5A, kurzschlußfest

Pin Nr.	Belegung
1	Ausgang 1
2	Ausgang 2
3	Ausgang 3
4	Ausgang 4
5	Ausgang 5
6	Ausgang 6
7	Ausgang 7
8	Ausgang 8
9	Ausgang 9
10	Ausgang10
11	Ausgang11
12	Ausgang12
13	Ausgang13
14	Ausgang14
15	Ausgang15
16	Ausgang16
17-19	GND
20	Eingang 1
21	Eingang 2
22	Eingang 3
23	Eingang 4
24	Eingang 5
25	Eingang 6
26	Eingang 7
27	Eingang 8
28	Eingang 9
29	Eingang 10
30	Eingang 11
31	Eingang 12
32	Eingang 13
33	Eingang 14
34	Eingang 15
35	Eingang 16
36	GND
37-40	+11,4...32V

Bestückungsplan

Platinennummer 06 14 98



6.12 Dokumentation: Trackball für LSTEP

Der Trackball für die LSTEP-xx/2 wurde entwickelt, um sehr feine manuelle Bewegungen auszuführen. Aktivieren kann man den Trackball durch einschalten des Joystick. Für größere Bewegungen verwendet man den Joystick, für kleine Bewegungen den Trackball. Sinnvoll ist, eine LSTEP mit Display zu verwenden, da die Tastenfunktionen im Display angezeigt werden.

Funktionen:

Der Trackball verfügt über drei zusätzliche Tasten.

1. Mit der linken und mittleren Taste läßt sich der Trackball-Faktor verändern.
2. Mit der rechten Taste lassen sich die Achsen X und Y einzeln für den Trackball sperren.

zu 1. Der Trackball-Faktor gibt an, wieviel Motorinkremente bei einem Trackball-Impuls ausgegeben werden. Die Grundeinstellung ist 1, d.h. 1 Impuls = 1 Motorinkrement. Mit der linken Taste kann man die Einstellung bis auf Faktor = 0,05 verkleinern, mit der mittleren Taste bis auf Faktor = 9,9 vergrößern. Drückt man die linke und mittlere Taste gleichzeitig gilt wieder die Grundeinstellung Faktor = 1. Der eingestellte Faktor wird immer für kurze Zeit im Display angezeigt.

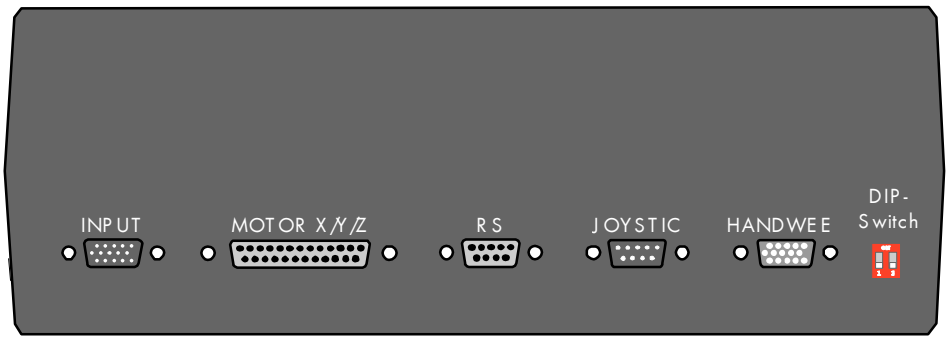
zu 2. Da es mit dem Trackball sehr schwer ist nur eine Achse allein zu bewegen, kann man mit der rechten Taste die Achsen abwechselnd sperren und wieder freigeben. Auch dies wird nach dem Tastendruck kurz im Display angezeigt.

Hinweis:

Über den Befehl " Trackball Back Lash " kann man für jede Achse ein Umkehrspiel einstellen, damit die Mechanik bei einem Richtungswechsel exakt der Trackballbewegung folgt. Weitere Informationen dazu finden sie in der Dokumentation Kapitel 4 / Befehlssatz LSTEP oder im Kapitel 9 / Anhang LSTEP_API.

7 Anhang ECO-STEP (ECO-DRIVE, ECO-MOT)

7.1 Die Rückwand der ECO-STEP



7.2 Steckerbelegung

7.2.1 Der Motoranschluß X/Y/Z

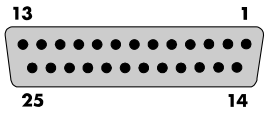


Abb.: Der Motoranschlus (25 Pol Sub-D Buchse)

Pin	Belegung	Pin	Belegung
1	Motor X, Phase 1 +	14	Motor Z, Phase 1 +
2	Motor X, Phase 1 -	15	Motor Z, Phase 1 -
3	Motor X, Phase 2 +	16	Motor Z, Phase 2 +
4	Motor X, Phase 2 -	17	Motor Z, Phase 2 -
5	Motor Y, Phase 1 +	18	Endschalter Y Nullpunkt
6	Motor Y, Phase 1 -	19	Endschalter Y Endlage
7	Motor Y, Phase 2 +	20	Endschalter Z Nullpunkt
8	Motor Y, Phase 2 -	21	Endschalter Z Endlage
9	Endschalter X Nullpunkt	22	+5V
10	Endschalter X Endlage	23	+12V
11	+ Versorgungsspannung Endstufe	24	GND
12	+ Versorgungsspannung Endstufe	25	GND
13	+ Versorgungsspannung Endstufe	Gehäuse	GND

7.2.2 Der Spannungsanschluß

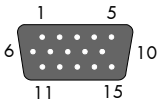


Abb.: Der Spannungsanschluß (15 Pol Sub-HD Stecker)

Pin	Belegung	Pin	Belegung
1,2	GND	14,15	+24V DC Geregelt

7.2.3 ST 4; 15-pol HD-Sub-Buchse: Koaxtrieb

Pin	Belegung
1	Analog VCC (+5V)
2	+5V
3	A+, X-Achse
4	A-, X-Achse
5	B+, X-Achse
6	B-, X-Achse
7	TTL-Eingang Auflösung
8	TTL_EingangSnap-Shot
9	Analog GND
10	Analog GND
11	C+, Y-Achse
12	C-, Y-Achse
13	D+, Y-Achse
14	D-, Y-Achse
15	Nc
Gehäuse	Schirm

7.2.4 ST 2: 9-pol D-Sub-Stecker: Joy-Stick, Stop, Snap-Shot

Pin	Belegung	Bemerkung
1	VAGND	Analog GND
2	/Joy-Stick ein	TTL, Pull Up = 4,7 kOhm
3	Joy-Stick X	
4	Joy-Stick Y	
5	Joy-Stick Z	
6	Snap-Shot	TTL, Pull Up = 4,7 kOhm
7	/Stop	TTL, Pull Up = 4,7 kOhm
8	VAREF	5V Analoge Referenzspspannung
9	VAREF	5V Analoge Referenzspspannung
Gehäuse	GND	

(Hinweis: Die Anschlüsse 3-5: Joy-Stick X,Y,Z und St4, Koaxtrieb: Achse X und Y können nur alternativ verwendet werden.)

7.2.5 ST3, 9-pol D-Sub-Buchse: RS 232-Schnittstelle

Pin	Belegung
1	nc.
2	RXD
3	TXD
4	GND
5	GND
6	+5V
7	RTS
8	CTS
9	+5V

7.2.6 St6, 10-pol. Pfostenleiste, D-Sub-Belegung: CAN-Bus

Pin	Belegung
1	NC
2	CAN L
3	CAN GND
4	NC
5	CAN Schirm (GND)
6	CAN GND
7	CAN H
8	NC
9	CAN V+ (J1 gesteckt: +12V)
10	NC
Gehäuse	Schirm

7.2.7 ST 8, 26-pol-Pfostenleiste: Anschluß für Frontplattenverbinder

Pin	Belegung
1,2	GND
3	RS
4	R, /WR
5	/E
6	DB 0
7	DB 1
8	DB 2
9	DB 3
10	DB 4
11	DB 5
12	DB 6
13	DB 7
14	/STOP
15	/Joy-Stick ein
16	/Clear X
17	/Clear Y
18	/Res in
19	VAGND
20	Speed
21	/CLR Z
22	+12 V
23	-12 V
24	Varef
25,26	+5V

7.3 Jumperbelegung

Bezeichnung	Funktion
J1	Gesteckt: CAN V+ = +12V
J2.1	Gesteckt: Varef von Präzisionsspannungsquelle
J2.2	Gesteckt: Varef von +5V

7.4 Belegung der DIP-Switches

Abb.: Die DIP-Switches der ECO-STEP

Schalter 1	ON	➔	Firmware update eingeschaltet
	OFF	➔	Firmware update ausgeschaltet
Schalter 2	ON	➔	Multicontrol-Befehlssatz
	OFF	➔	Standard-Befehlssatz

7.5 Technische Daten

Netzanschluss:	Tischnetzteil / AC INPUT: 110V - 240V 1,5 A 47-63 Hz DC OUTPUT: +24V ---- 3A
Max. Netzausfalldauer:	< 50ms bei Netzausfall (<0,77 * UN) schaltet die LSTEP auf Reset
Max. Motordrehzahl:	15 U/sec. bei 200-schrittigem Motor
Max. Motorstrom:	1,25A je Motorphase
Max. Motorspannung:	24V
Schrittauflösung:	max. 50.000 Schritte/Umdrehung bei 200schrittigem Motor
Baudrate:	57,6 Kbd
Umgebungsbedingungen:	
Lufttemperatur bei Betrieb:	15 ... 40 Grad C
Lufttemperatur ausser Betrieb:	0 ... 43 Grad C
Luftfeuchtigkeit bei Betrieb	8 ... 80 % bei 31° / Maximal 50% bei 40°
Luftfeuchtigkeit ausser Betrieb	0 ... 80 %
Abmessungen B • T • H:	
	ohne Anzeige 245mm • 185mm • 90mm
	mit Anzeige 245mm • 225mm • 90mm
Gewicht:	3,5kg

ECO-DRIVE

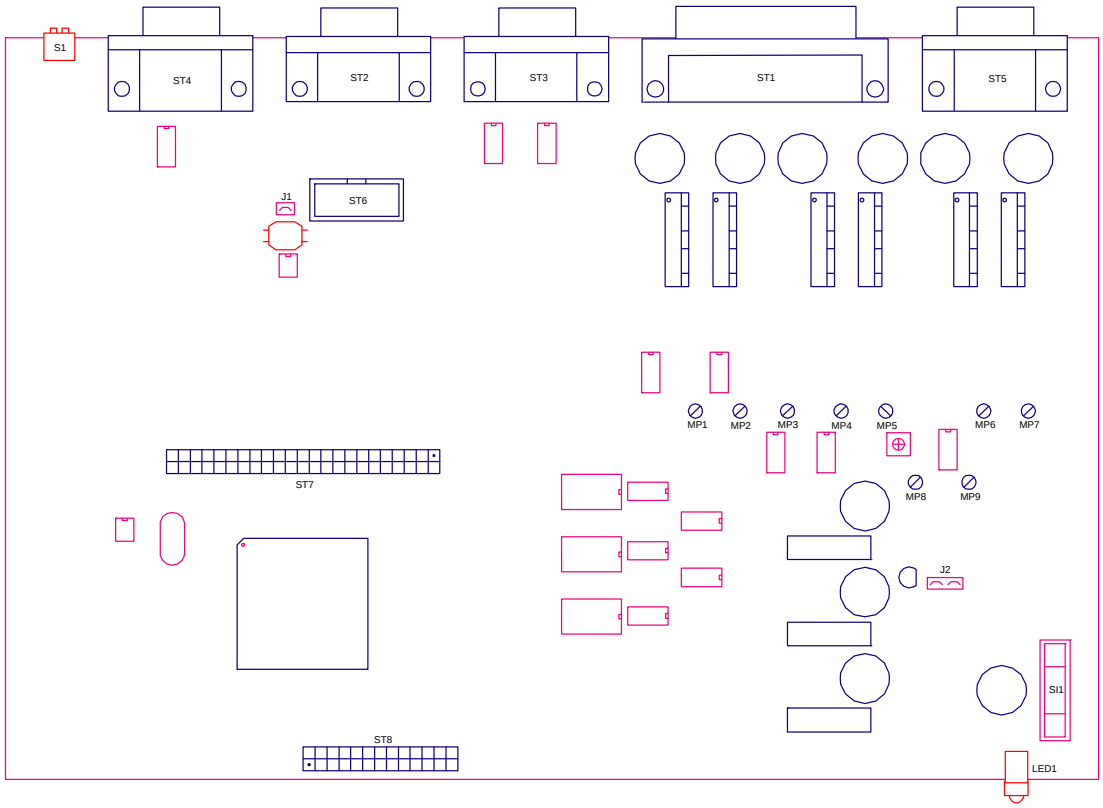
Bei der ECO-DRIVE ist nur die X + Y-Achse bestückt, und die Geschwindigkeit ist auf 5 U/s begrenzt. Sie wird eingesetzt für Ritzel-Zahnstangen Koordinatentische mit großer Steigung (28mm / Umdrehung)

ECO-MOT

Die ECO-MOT ist eine Einachs-Steuerung, bei der nur die Z- Achse bestückt ist. Sie wird für Focus-Antriebe eingesetzt.

Achtung! Ist die Steuerung auf den Register-Befehlsatz eingestellt, muß man bei den API-Befehlen die Z-Achse ansprechen, außer bei SetVel und SetAccel, dort wird der X-Wert übernommen.

7.6 Die Platinenansicht



7.7 Das Netzteil

Die Steuerung wird mit einem externen Netzteil geliefert.

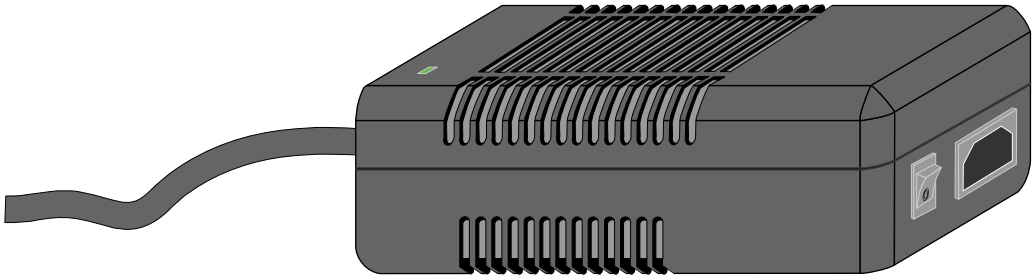


Abb.: Das externe Netzteil

7.7.1 Technische Daten des Netzteils

Netzanschluss:	Weitbereichseingang 100 bis 240V~ / 1,5A, 47-63Hz
Ausgang:	+24V ⎓ 3A

8 Hardware Dokumentation der LSTEP-PCI

8.1 Jumper / PCI

Bezeichnung	Funktion wenn Jumper gesteckt
J1.1	RS 232 - Schnittstelle St2,Pin9 = +5V
J1.2	RS 232 - Schnittstelle St2,Pin9 = +12V
J2	CAN-BUS-Stecker St7,Pin9 = +12V
J3	Soll St11,Pin18 und St10,Pin5 Digital-I/O sein, dann muß OP1 entfernt werden und J3 gesteckt sein.
J4	VPP-Eingang Controller ist auf +12V gelegt (für Flash-Programmierung)
J5.1	PCI-Baustein wird durch den Inhalt des EEPROMS (IC21) gebootet.
J5.2	PCI-Baustein benutzt seine Standard Vendor- und ID-Nummer (bootet intern)

0-Ohm Widerstände LSTEP-PCIcompact	
Bezeichnung / (R ...)	**Funktion wenn Widerstand bestückt**
0E*1 / 12; 13; 14; 15; 16; 65	NPN – Endschalter (Vorzugsbestückung)
0E*2 / 32; 34; 36; 37; 38; 97	PNP – Endschalter
/ 165	22V Verbindung zu den Endstufen OP´s

8.2 Schalter / PCI und PCIcompact

Bezeichnung	Funktion wenn Schalter an
S1	Nach Reset geht die Steuerung in den Bootstrapmode
S2	Reset aktiv

8.3 Lötbrücken / PCI

Lötbrücke (LB)	Funktion geschlossen/ Anmerkung
1	Abschlußwiderstand CAN-Bus (120 Ohm) an St7,Pin 2 und 7
2	Versorgungsspannung Endstufe X (dient zur Inbetriebnahme)
3	Versorgungsspannung Endstufe Y (dient zur Inbetriebnahme)
4	Versorgungsspannung Endstufe Z (dient zur Inbetriebnahme)
5	Zum Trennen des PT-100-Sensorverstärker von St11,Pin6. Dieser kann dann als Analog- oder Digitaleingang verwendet werden.
6.1	EEPROM IC21 wird mit +5V versorgt
6.2	EEPROM IC21 wird mit +3,3V versorgt.
12.1	Ausgangsspannung an St11,Pin18 und St10,Pin5 (Aout) = +/- 10V
12.2	Ausgangsspannung an St11,Pin18 und St10,Pin5 (Aout) = 0...10V

Lötbrücken / PCIcompact

Lötbrücke (LB)	Funktion geschlossen/ Anmerkung
5	Abschlußwiderstand CAN-Bus (120 Ohm) an St7,Pin 2 und 7
4.1	RS 232 - Schnittstelle St2,Pin9 = +5V
4.2	RS 232 - Schnittstelle St2,Pin9 = +12V
2	CAN-BUS-Stecker St7,Pin9 = +12V
3	Zum Trennen des PT-100-Sensorverstärker von St11,Pin6. Dieser kann dann als Analog- oder Digitaleingang verwendet werden.
1	Soll St11,Pin18 und St10,Pin5 Digital-I/O sein, dann muß OP1 entfernt werden und J3 gesteckt sein.
6.1	Ausgangsspannung an St11,Pin18 und St10,Pin5 (Aout) = +/- 10V
6.2	Ausgangsspannung an St11,Pin18 und St10,Pin5 (Aout) = 0...10V

8.4 LED´s PCI

Bezeichnung	Funktion
LED1	Endstufe X aktiv
LED2	Endstufe Y aktiv
LED3	Endstufe Z aktiv

LED´s / PCIcompact

Bezeichnung	Funktion
LED 3	Endstufe X aktiv
LED 1	Endstufe Y aktiv
LED 2	Endstufe Z aktiv

8.5 Stecker

8.5.1 ST1, 9-pol D-Sub-Stecker: Joy-Stick, Stop, Snap-Shot / PCI und PCIcompact

Pin Nr	Belegung	Bemerkung
1	VAGND	Analog GND
2	/Joy-Stick ein	TTL, Pull Up = 4,7 kOhm, RC-Filter 470 Ohm/100nF
3	Joy-Stick X	RC-Filter 10kOhm/10nF
4	Joy-Stick Y	RC-Filter 10kOhm/10nF
5	Joy-Stick Z	RC-Filter 10kOhm/10nF
6	Snap-Shot	TTL, Pull Up = 4,7 kOhm, RC-Filter 470 Ohm/100nF
7	/Stop	TTL, Pull Up = 4,7 kOhm, RC-Filter 470 Ohm/100nF
8	VAREF	5V Analoge Referenzspspannung
9	VAREF	5V Analoge Referenzspspannung
Gehäuse	GND	

Hinweis: Die Anschlüsse 3-5: Joy-Stick X,Y,Z sind identisch mit ST 11, Pin`s 24,12 und 25.

8.5.2 ST2, 10-pol Pfostenleiste mit D-Sub-Belegung: RS 232-Schnittstelle / PCI und PCIcompact

Pin Nr.	Belegung
1	nc.
2	RXD
3	TXD
4	GND
5	GND
6	+5V
7	RTS
8	CTS
9	J1.1 gesteckt: +5V; J1.2 gesteckt: +12V

8.5.3 St7, 10-pol. Pfostenleiste, D-Sub-Belegung: CAN-Bus/ PCI und PCIcompact

Pin Nr.	Belegung
1	NC
2	CAN L
3	CAN GND
4	NC
5	CAN Schirm (GND)
6	CAN GND
7	CAN H
8	NC
9	CAN V+ (J2 gesteckt: +12V)
10	NC

8.5.4 St5, 8-pol Pfostenleiste Meßpunkt 1-8 / PCI und PCIcompact

Pin Nr	Bezeichnung	Funktion
1	MP1	Meßpunkt: Port 8.0 des Controllers (auch intern verwendet)
2	MP2	Meßpunkt: Port 8.1 des Controllers (auch intern verwendet)
3	MP3	Meßpunkt: Port 8.2 des Controllers (auch intern verwendet)
4	MP4	Meßpunkt: Port 8.3 des Controllers (auch intern verwendet)
5	MP5	Meßpunkt: Port 8.4 des Controllers
6	MP6	Meßpunkt: Port 8.5 des Controllers
7	MP7	Meßpunkt: Port 8.6 des Controllers
8	MP8	Meßpunkt: Port 8.7 des Controllers

8.5.5 ST3, 25-pol D-Sub-Buchse: Motor- und Endschalteranschüsse / PCI und PCIcompact

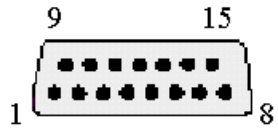
Pin Nr	Belegung
1	Motor X, Phase 1 +
2	Motor X, Phase 1 -
3	Motor X, Phase 2 +
4	Motor X, Phase 2 -
5	Motor Y, Phase 1 +
6	Motor Y, Phase 1 -
7	Motor Y, Phase 2 +
8	Motor Y, Phase 2 -
9	Endschalter X Nullpunkt
10	Endschalter X Endlage
11	+ Versorgungsspannung Endstufe
12	+ Versorgungsspannung Endstufe
13	+ Versorgungsspannung Endstufe
14	Motor Z, Phase 1 +
15	Motor Z, Phase 1 -
16	Motor Z, Phase 2 +
17	Motor Z, Phase 2 -
18	Endschalter Y Nullpunkt
19	Endschalter Y Endlage
20	Endschalter Z Nullpunkt
21	Endschalter Z Endlage
22	+5V
23	+12V
24	GND
25	GND
Gehäuse	GND

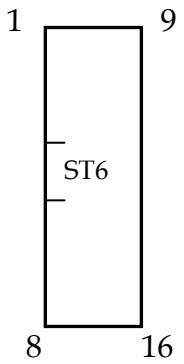
Wird die Betriebsspannung für die Endstufe an ST3 eingespeist, so ist auf eine ausreichende Stromtragfähigkeit der Stecker und Kabel zu achten (insbesondere bei Flachbandkabel).

8.5.6 St6, 16-pol Pfostenleiste (D-Sub-Zählweise): TTL-Gebereingänge / PCI und PCIcompact

Alle Eingänge haben TTL-Pegel und Pull-up-Widerstände 4,7kOhm gegen +5V.
St6 und St8 dürfen nur alternativ verwendet werden. Die maximale Zählfrequenz beträgt 2,5 Mflanken = 625 KHz.

Pin Nr	Bezeichnung	Funktion
1	Ph1A	Incrementalgeber 1, Spur A
2	Ph1B	Incrementalgeber 1, Spur B
3	Ph1Z	Incrementalgeber 1, Spur Z (Referenzsignal)
4	Ph2A	Incrementalgeber 2, Spur A
5	Ph2B	Incrementalgeber 2, Spur B
6	Ph2Z	Incrementalgeber 2, Spur Z (Referenzsignal)
7	GND	
8	GND	
9	Ph3A	Incrementalgeber 3, Spur A
10	Ph3B	Incrementalgeber 3, Spur B
11	Ph3Z	Incrementalgeber 3, Spur Z (Referenzsignal)
12	+5V	
13	+5V	
14	+12V	
15	+12V	
16	nc	





8.5.7 St8, 16-pol-Pfostenleiste (normalzählweise): Geber-Aufsatzkarte / PCI

Pin Nr	Bezeichnung	Funktion /Bemerkung
1	Ph1A	Incrementalgeber 1, Spur A
2	Ph1B	Incrementalgeber 1, Spur B
3	Ph1Z	Incrementalgeber 1, Spur Z (Referenzsignal)
4	Ph2A	Incrementalgeber 2, Spur A
5	Ph2B	Incrementalgeber 2, Spur B
6	Ph2Z	Incrementalgeber 2, Spur Z (Referenzsignal)
7	ClkIn	TTL-Clocksignal von T6, IC7/Pin66
8	Ph1A	Incrementalgeber 3, Spur A
9	Ph1B	Incrementalgeber 3, Spur B
10	Ph1Z	Incrementalgeber 3, Spur Z (Referenzsignal)
11	/ERRX	TTL-Eingang Errorsignal X (aktiv L)
12	/ERRY	TTL-Eingang Errorsignal Y (aktiv L)
13	/ERRZ	TTL-Eingang Errorsignal Z (aktiv L)
14	CSAD	CS-Signal 4, IC7,Pin3: Für AD-Wandler (aktiv L)
15, 16	nc	

8.5.8 St11, 26-pol Pfostenleiste, D-Sub Zählweise: Multifunktionsport / PCI und PCIcompact

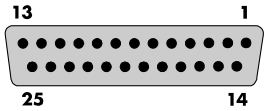


Abb.: Der Multifunktionsport (25 Pol Sub-D Buchse)

Bedingt durch die Funktionsvielfalt sind die Pins des Multifunktionsport *(MFP)* teilweise mehrfach belegt. Je nach Ausstattung der Steuerung bedeutet dies, dass jeweils nur ein Signalaus- bzw. eingang auf einem Pin des MFP anliegt.
Die gewünschte Funktionalität muß bei der Bestellung mit angegeben werden.
Standart ist: Trigger, Snapshot, und Stopeingang

Pin Nr	Bezeichnung	Bemerkung
1	Takt X	Takt Ein- oder Ausgang, 10kOhm gegen +5V, RC-Glied 470Ohm/220pF
2	V/R X	V/R Ein- oder Ausgang, 10kOhm gegen +5V, RC-Glied 470Ohm/220pF
3	Takt Y	Takt Ein- oder Ausgang, 10kOhm gegen +5V, RC-Glied 470Ohm/220pF
	Tigger out 2	Standard: TTL-Pegel / I_{max} = 1,6 mA
4	V/R Y	V/R Ein- oder Ausgang, 10kOhm gegen +5V, RC-Glied 470Ohm/220pF
5	Takt Z	Takt Ein- oder Ausgang, 10kOhm gegen +5V, RC-Glied 470Ohm/220pF
6	Ain 10 Channel 10	Nur benutzbar wenn Verbindung J5 entfernt ist (St10,Pin6 und 7 ist dann inaktiv): Standard: TTL-Eingang, 4,7kOhm Pull-Up, RC-Filter 10kOhm/100nF, Option: Analogeingang 0...5V
7	Ain 8 Channel 8	Standard: TTL-Eingang, 4,7kOhm Pull-Up, RC-Filter 10kOhm/100nF Option: Analogeingang 0...5V (=St10,Pin3)
8	Ain 6 Channel 6	Standard: TTL-Eingang, 4,7kOhm Pull-Up, RC-Filter 10kOhm/100nF Option: Analogeingang 0...5V (=St10,Pin1)
9	- 12V	
10	/Joystick ein	Entspricht St1,Pin2: TTL, Pull Up = 4,7 kOhm, RC-Filter 470 Ohm/100nF
11	VAGND	
12	Ain 1 /Channel 1	RC-Filter 10kOhm/100nF **Joystick Y**
13	VAREF	+5V Referenzspannung
14	V/R Z	V/R Ein- oder Ausgang, 10kOhm gegen +5V, RC-Glied 470Ohm/220pF
15	Tigger out	HCMOS-Ausgang: I,max = 1,6 mA
16	GND	
17	+5V	
18	Analog Out Channel 0	Standard: Analogausgang 0...10V bzw. +/-10V je nach LB12, Ri,min = 1kOhm, Option: Digital I/O (siehe Jumper 3) (=St10,Pin5)
19	Ain 9 Channel 9	Standard: TTL-Eingang, 4,7kOhm Pull-Up, RC-Filter 10kOhm/100nF Option: Analogeingang 0...5V (=St10,Pin4)
20	Ain 7 Channel 7	Standard: TTL-Eingang, 4,7kOhm Pull-Up, RC-Filter 10kOhm/100nF Option: Analogeingang 0...5V (=St10,Pin2)
21	+12V	

22	SnapShot	Eingang: TTL, Pull Up = 4,7 kOhm, RC-Filter 470 Ohm/100nF
23	/Stop	Eingang: TTL, Pull Up = 4,7 kOhm, RC-Filter 470 Ohm/100nF
24	Ain 0 Channel 0	RC-Filter 10kOhm/100nF **Joystick X**
25	Ain 2 Channel 2	RC-Filter 10kOhm/100nF **Joystick Z**
26	Ain 3 Channel 3	Standard: TTL-Eingang, 4,7kOhm Pull-Up, RC-Filter 10kOhm/100nF Option: Analogeingang 0...5V

8.5.9 St10, 10-pol Pfostenleiste mit D-Sub-Belegung: Analog I/O / PCI und PCIcompact

Pin Nr	Belegung	Bemerkung
1	Analog In 1 Channel 6	Analog: 0...5V, 4,7kOhm gegen +5V, RC-Filter 10KOhm/100nF (=St11,Pin8)
2	Analog In 2 Channel 7	Analog: 0...5V, 4,7kOhm gegen +5V, RC-Filter 10KOhm/100nF (=St11,Pin20)
3	Analog In 3 Channel 8	Analog: 0...5V, 4,7kOhm gegen +5V, RC-Filter 10KOhm/100nF (=St11, Pin7)
4	Analog In 4 Channel 9	Analog: 0...5V, 4,7kOhm gegen +5V, RC-Filter 10KOhm/100nF (=St11,Pin19)
5	Analog Out Channel 0	0...10V oder +/-10V, R,Last >=1kOhm Ri= ca. 100 Ohm (=St11,Pin18)
6, 7	PT 100 Channel 10 Temperaturfühleranschluß	Meßstrom = 10 mA, LB 5 muß geschlossen sein, St11, Pin6 ist nicht verwendbar)
8	GND	
9	VAREF = +5V / 1A	Ausgang
10	NC	

8.5.10 ST4, 4-pol PC-Netzteilstecker: Motorspannungsversorgung/ PCI und PCIcompact

Pin Nr	Belegung	Bemerkung
1	+Um	Motorspannungsversorgung: Bei Verwendung des PC-Netzteils = 12V, bei externem Netzteil = 11,4....48V (48V=max. nur geregeltes Netzteil verwenden)
2,3	GND	
4	NC	

8.5.11 St 9, 46-pol Buchsenleiste: Systembus (für Erweiterungsmodule) / PCI

Pin Nr.	Belegung
1	D0
2	D1
3	D2
4	D3
5	D4
6	D5
7	D6
8	D7
9	D8
10	D9
11	D10
12	D11
13	D12
14	D13
15	D14
16	D15
17	A0
18	A1
19	A2
20	A3
21	A4
22	A5
23	A6
24	A7
25	A8
26	A9
27	A10
28	A11
29	A12
30	A13
31	A14
32	A15
33	A16
34	/CS0
35	/RD
36	/WR
37	-12V
38	+12V
39	+5V
40	GND
41	Digital I/O 1, Interruptfähig
42	Reset Out
43	Analog/ Digital Eingang 1
44	Analog/ Digital Eingang 2
45	Analog/ Digital Eingang 3
46	Digital I/O 2 (Derzeit intern belegt: VPP Flash ein)

8.5.12 St 8 / 50-pol Buchsenleiste: Für Sin.- Cos.- Encoderauswertung / PCIcompact

Pin Nr.	Belegung
1	D0
2	D1
3	D2
4	D3
5	D4
6	D5
7	D6
8	D7
9	D8
10	D9
11	D10
12	D11
13	GND
14	A1
15	A2
16	A3
17	A6
18	A7
19	A8
20	A9
21	A10
22	A11
23	A12
24	A13
25	A14
26	A15
27	A16
28	CSO
29	/RD
30	/WR
31	-12V
32	+12V
33	+5V
34	GND
35	P7.4
36	/RST
37	Takt X
38	U/D X
39	Takt Y
40	U/D Y
41	Takt Z
42	U/D Z
43	CIKin
44	/Ref X
45	/Ref Y
46	/Ref Z
47	/Err X
48	/Err Y
49	/Err Z
50	CSAD

8.5.13 St12, PCI-Bus / PCI und PCIcompact

Es werden nur Pin´s gelistet, die verwendet werden.

Bezeichnung	Pin Nr.
AD0	A58
AD1	B58
AD2	A57
AD3	B56
AD4	A55
AD5	B55
AD6	A54
AD7	B53
AD8	B52
AD9	A49
AD10	B48
AD11	A47
AD12	B47
AD13	A46
AD14	B45
AD15	A44
AD16	A32
AD17	B32
AD18	A31
AD19	B30
AD20	A29
AD21	B29
AD22	A28
AD23	B27
AD24	A25
AD25	B24
AD26	A23
AD27	B23
AD28	A22
AD29	B21
AD30	A20
AD31	B20
C/BE0	A52
C/BE1	B44
C/BE2	B33
C/BE3	B26
/INTA	A6
PAR	A43
/SERR	B42
/PERR	B40
/STOP	A38
/DEVSEL	B37
/TRDY	B35
/IRDY	B35
/FRAME	A34
IDSEL	A26
/REQ	B18
/GNT	A17

CLK	B16
/RESET	A15
/PRSNT1	B9
/PRSNT2	B11
PCI-VIO	B19,B59,A10,A16,A59
-12V	B1
+12V	A1
+5V	A5,A8,A61,A62,B5,B6,B61,B62
GND	A:18,24,30,35,37,42,48,56 B:3,15,17,22,28,34,38,46,49,57

8.5.14 St14 / 10-pol Pfostenleiste mit D-Sub-Belegung: Endschalter

Pin Nr.	Belegung	Bemerkung : Bestückungsvarianten
1	Endschalter 0-Position X	NPN = R15 bestückt / PNP = R37 bestückt
2	Endschalter End-Position X	NPN = R65 bestückt / PNP = R97 bestückt
3	Endschalter 0-Position Y	NPN = R16 bestückt / PNP = R38 bestückt
4	Endschalter End-Position Y	NPN = R14 bestückt / PNP = R36 bestückt
5	Endschalter 0-Position Z	NPN = R12 bestückt / PNP = R32 bestückt
6	Endschalter End-Position Z	NPN = R13 bestückt / PNP = R34 bestückt
7	+5V	
8	+12V	
9	GND	
10	nc	

- Es darf nur ein Widerstand pro Endschaltereingang bestückt werden.
- Die Grundbestückung ist für NPN Endschalter ausgelegt.

8.5.15 St 9 / 40-pol Pfostenleiste mit DSub Belegung: 16 digitale I/O´s / PCIcompact

Pin Nr.	Belegung
1	Ausgang 1
2	Ausgang 2
3	Ausgang 3
4	Ausgang 4
5	Ausgang 5
6	Ausgang 6
7	Ausgang 7
8	Ausgang 8
9	Ausgang 9
10	Ausgang 10
11	Ausgang 11
12	Ausgang 12
13	Ausgang 13
14	Ausgang 14
15	Ausgang 15
16	Ausgang 16
17	GND
18	GND
19	GND
20	Eingang 1
21	Eingang 2
22	Eingang 3
23	Eingang 4
24	Eingang 5
25	Eingang 6
26	Eingang 7
27	Eingang 8
28	Eingang 9
29	Eingang 10
30	Eingang 11
31	Eingang 12
32	Eingang 13
33	Eingang 14
34	Eingang 15
35	Eingang 16
36	GND
37	+24V
38	+24V
39	+24V
40	+24V

8.5.16 ST15 / 2-pol Stecker: 24V Spannungsversorgung für digitale I/O´s / PCIcompact

Pin Nr.	Belegung
1	+24V
2	GND

8.6 Meßpunkte PCI / PCI Compact

Meßpunkt /PCI	Bezeichnung	Funktion
MP1	GND	
MP2	MCOSX	Meßpunkt Motorstrom X-Achse, Phase 2 (cos)
MP3	MCOSY	Meßpunkt Motorstrom Y-Achse, Phase 2 (cos)
MP4	MCOSZ	Meßpunkt Motorstrom Z-Achse, Phase 2 (cos)
MP5	MSINX	Meßpunkt Motorstrom X-Achse, Phase 1 (sin)
MP6	MSINY	Meßpunkt Motorstrom Y-Achse, Phase 1 (sin)
MP7	MSINZ	Meßpunkt Motorstrom Z-Achse, Phase 1 (sin)
MP8	GND	
MP9	+3,3V	+3,3V Spannungsversorgung
MP10	Sz.gen. 20kHz	Signal Dreieckgenerator
MP11	GND	

Meßpunkt / PCIcompakt Nr.	Bezeichung	Funktion
MP10	GND	
MP3	MCOSX	Meßpunkt Motorstrom X-Achse, Phase 2 (cos)
MP1	MCOSY	Meßpunkt Motorstrom Y-Achse, Phase 2 (cos)
MP6	MCOSZ	Meßpunkt Motorstrom Z-Achse, Phase 2 (cos)
MP4	MSINX	Meßpunkt Motorstrom X-Achse, Phase 1 (sin)
MP2	MSINY	Meßpunkt Motorstrom Y-Achse, Phase 1 (sin)
MP5	MSINZ	Meßpunkt Motorstrom Z-Achse, Phase 1 (sin)
MP7	GND	
MP8	+3,3V	+3,3V Spannungsversorgung
MP11	Sz.gen. 20kHz	Signal Dreieckgenerator

8.7 Sicherungen PCI / PCIcompact

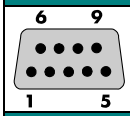
Nr.	Bemerkung
SI 1 PCI und PCIcompact	Sichert St4,Pin1 und St3,Pin11-13 (Versorgungsspannung Endstufen) ab. Wert: max. F 5A. Beim Absichern die Stromtragfähigkeit der verwendeten Kabel (insbesondere wenn an ST3 mit Flachbandkabel versorgt wird) beachten.
SI 2 PCIcompact	Sichert die 24V Eingangsspannung für die digitalen Ein-Ausgänge ab.

8.8 Geberadapterkarte für LSTEP PCI mit analogen Ausgängen oder PCIcompact

Mit der Geberadapterkarte 06 09 04 können 3 Geber adaptiert werden und 2 Analoge Ausgänge mit 0...10V oder +/-10V gesteuert werden. Beim Geberinterface stehen Differenzeingänge für Sinus, Cosinus und Referenzsignal zur Verfügung. Je nach Bestückung können 1Vss (z. B. bei optischen Gebern), 5Vss (z. B. bei MR-Sensoren) adaptiert werden.
Die Analogausgänge bestehen aus einem 2-fach 8-Bit-Wandler der durch OP9 verstärkt wird. Ist J1.1 bzw. J2.1 gesteckt, dann ist der Ausgangsbereich +/-10V. Ist J1.2 bzw. 2.2 gesteckt, dann ist der Ausgangsspannungsbereich 0...10V. L1 und L2 sollen Schwingneigung bei kapazitiven Lasten vermeiden.
Die Ausgänge sind mit +/- 5mA belastbar. Der Innenwiderstand beträgt ca. 100 Ohm.

Lötbrücken	Funktion (geschlossen)
LB1	+12V an St1,Pin 4
LB2	+12V an St2,Pin 4
LB3	+12V an St3,Pin 4
LB4	120 Ohm Abschlußwiderstand St1,Pin 6 und 1 (sin)
LB5	120 Ohm Abschlußwiderstand St1, Pin 3 und 8 (cos)
LB6	120 Ohm Abschlußwiderstand St1, Pin 5 und 9 (ref)
LB7	120 Ohm Abschlußwiderstand St2,Pin 6 und 1 (sin)
LB8	120 Ohm Abschlußwiderstand St2, Pin 3 und 8 (cos)
LB9	120 Ohm Abschlußwiderstand St2, Pin 5 und 9 (ref)
LB10	120 Ohm Abschlußwiderstand St3,Pin 6 und 1 (sin)
LB11	120 Ohm Abschlußwiderstand St3, Pin 3 und 8 (cos)
LB12	120 Ohm Abschlußwiderstand St3, Pin 5 und 9 (ref)
LB 13	5,1kOhm Pull Up Widerstand an St5,Pin 41 (Interrrupt AD-Wandler)

Jumper	Funktion
1.1	Spannungsbereich an ANOut1 = +/- 10V
1.2	Spannungsbereich an ANOut1 = 0...10V
2.1	Spannungsbereich an ANOut2 = +/- 10V
2.2	Spannungsbereich an ANOut2 = 0...10V

Stecker	ST1, 1, 2, 3, Stecker 1(X),2(Y),3(Z) Geberstecker
	*Abb.: Der Geberanschluß (9 Pol Sub-D Buchse)*
Pin Nr.	**Funktion**
1	- Sin
2	GND
3	-Cos
4	wenn LB 1,2 oder 3 gesteckt sind, dann sind ST 1,2,3 = +12V
5	- Ref
6	+Sin
7	+5V
8	+Cos
9	+Ref
Spannungen: Geberinterface: 0,6...1,2Vss; MR-Interface max. 5Vss	

Pin Nr.	ST4, 16-pol Pfostenleiste PCI	
	Bezeichnung	Funktion / Bemerkung
1	Takt X	Taktsignal des Gebers X
2	U/D X	Richtungssignal des Gebers X
3	Takt Y	Taktsignal des Gebers Y
4	U/D Y	Richtungssignal des Gebers Y
5	Takt Z	Taktsignal des Gebers Z
6	U/D Z	Richtungssignal des Gebers Z
7	Clk	TTL-Clocksignal von T6
8	/Ref X	Referenzsignal Geber X
9	/Ref Y	Referenzsignal Geber Y
10	/Ref Z	Referenzsignal Geber Z
11	/ERRX	TTL-Errorsignal X (aktiv L)
12	/ERRY	TTL-Errorsignal Y (aktiv L)
13	/ERRZ	TTL-Errorsignal Z (aktiv L)
14	/CSAD	CS-Signal 4: Für AD-Wandler (aktiv L)
15,16	nc	

ST5: PCIcompact ST6: PCI 10-pol Pfostenleiste (D-Sub-Belegung): Analog Out	
Pin Nr.	Funktion
Pin Nr.	Funktion
1,2	GND
3	Ausgang 1
4	Ausgang 2
7	+5V
8	+12V
9	-12V

8.9 Beschreibung I / O - Karte für die LSTEP-PCI
(bei der PCIcompact sind die I/O´s mit auf der Platine)

Die 16 Ein- und 16 Ausgänge umfassende Karte ist für 46-pin Pfostenleisten-Busadapter geeignet. Der Formfaktor paßt auf die LSTEP-PCI (Rechtwinklige Anordnung von 46-pol-Leiste und I/O-Kabelabgang; Es wird kein weiterer Slot verbaut.) /CS, /RD und /WR-Signale von 65ns Länge sind ausreichend, während der Bus erst 25ns nach /RD floatet. Für einen C168 mit 20 Mhz werden keine Waitstates, early Read/Write, keinen ALE-Extender aber ein Floatextender mindestens benötigt.
Die Belegung von ST2 weicht von der Belegung des I/O-Steckers der LSTEP-PC-Karte ab. Grund: Bei einer Flachbandleitung können die einzelnen Adern jeweils nur mit einem Strom von 1A beaufschlagt werden. Bei LSTEP-PC wird die Versorungsspannung von außen zugeführt. D.h. die +11,4...32V-Leitung trägt den gesamten Strom, während die GND-Leitung nahezu unbelastet ist. Die +11,4...32V-Leitung ist deshalb 4-fach ausgeführt. Bei der vorliegenden Karte wird der Strom auf der Karte eingespeist (ST4). Dadurch wird die +11,4...32-V-Leitung nahezu nicht belastet, während die GND-Leitung als Rückleiter den gesamten Strom trägt. Sie wird deshalb hier 4-fach ausgeführt. Bei externer Stromversorgung muß die 11,4...32V-Versorgungsspannung unbedingt über ST4 eingespeist werden. Eine Einspeisung über ST2 ist unzulässig.

8.9.1 ST1: Belegung des 46-pin-Busadapters: PCI

Pin Nr.	Funktion
1-16	D0 – D15
18	A1
21 - 33	A4 - A16
34	/CS
35	/RD
36	/WR
39	+ 5V
40	GND
41	Interrupt: Low, wenn Ausgänge überlastet (Diag. =L); LB1 geschlossen: Pull up 10kOhm gegen +5V
42	/RSTOUT

8.9.2 2-pol Power Stecker für die Versorgung der Ein- und Ausgänge ST4/ PCI

Verwendet wird ein Phönix Mini-Combicon Grundgehäuse 2-polig.
Die Powerversorgung wird auf der Platine per „Einlötsicherung“ F4A (Hersteller: Wickmann) abgesichert. Da die Ausgänge kurzschlußfest sind, löst ein Kurzschluß am Ausgang nicht die Sicherung aus. LED1 (grün) zeigt an, daß Spannung hinter der Sicherung anliegt.

Pin Nr.	Funktion
1	+11,4...32V
2	0V

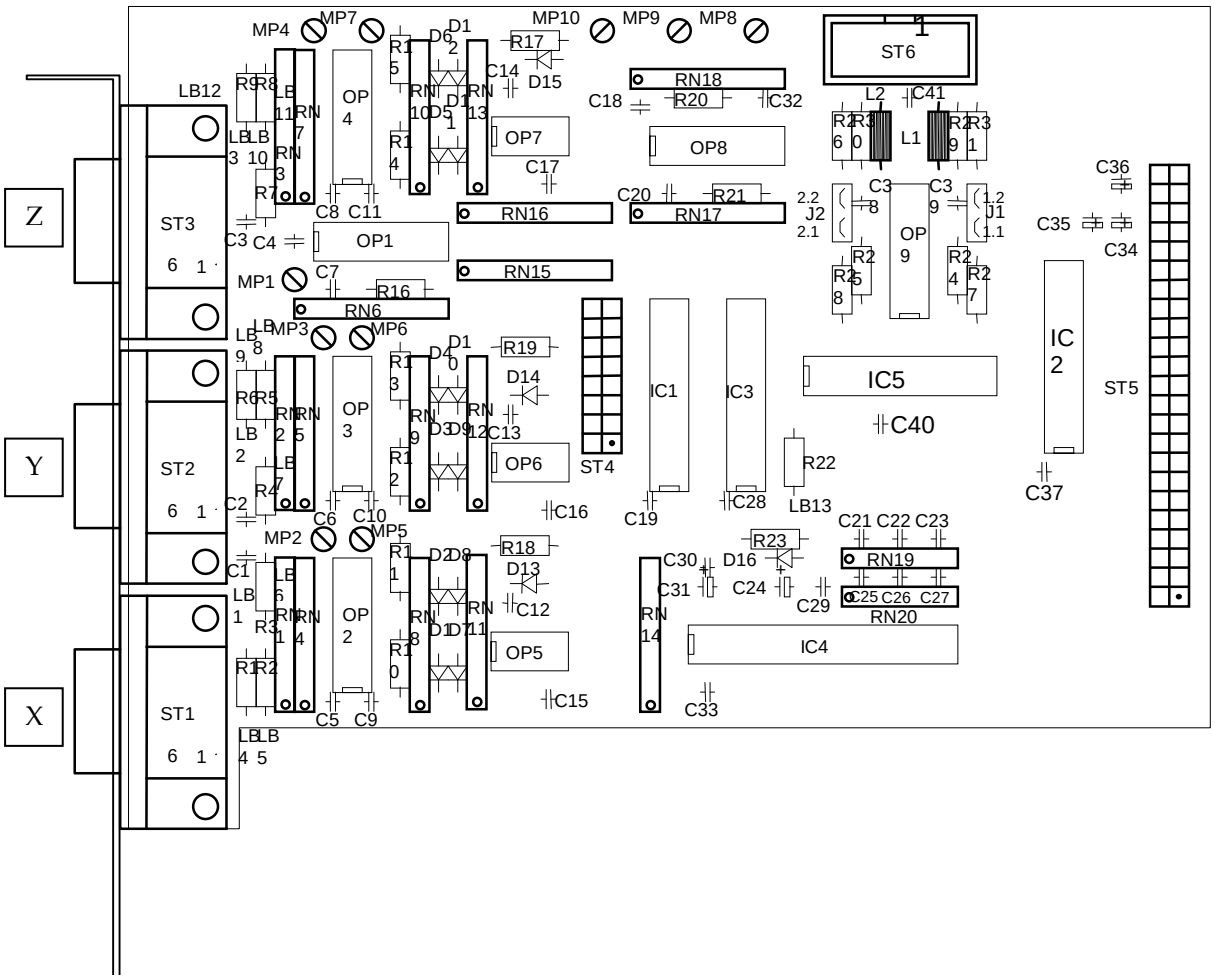
8.9.3 40-pol Pfostenleiste mit 37-pol D-Sub-Stecker-Belegung: 16 Eingänge, 16 Ausgänge ST2/ PCI

Eingänge: 0...2,7V = „L“, 7,5...32V = „H“, Ri = ca. 10kOhm
Ausgänge: Schaltend nach +Ub=11,4...32V, Imax = 0,5A, kurzschlußfest

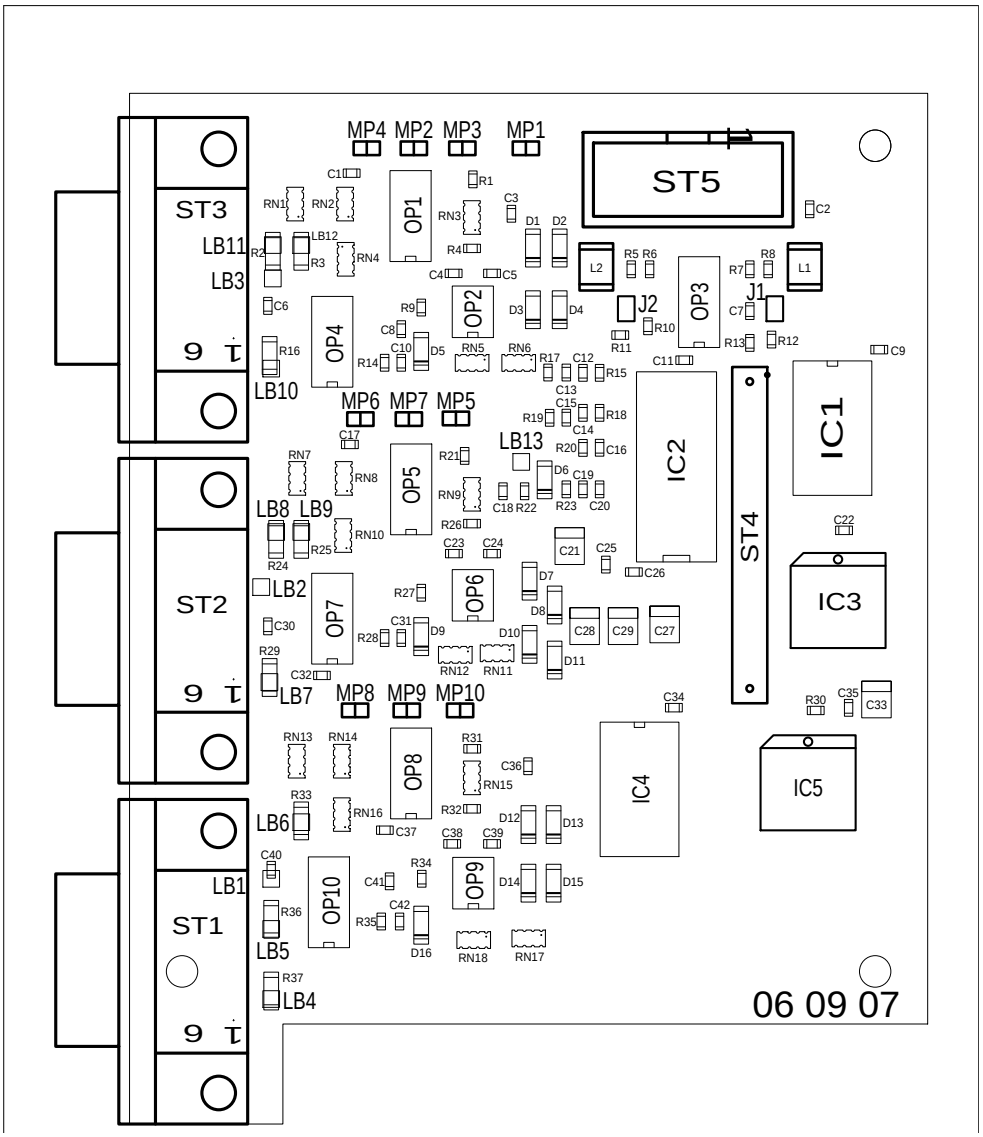
Pin Nr.	Belegung
1	Ausgang 1
2	Ausgang 2
3	Ausgang 3
4	Ausgang 4
5	Ausgang 5
6	Ausgang 6
7	Ausgang 7
8	Ausgang 8
9	Ausgang 9
10	Ausgang10
11	Ausgang11
12	Ausgang12
13	Ausgang13
14	Ausgang14
15	Ausgang15
16	Ausgang16
17-19	GND
20	Eingang 1
21	Eingang 2
22	Eingang 3
23	Eingang 4
24	Eingang 5
25	Eingang 6
26	Eingang 7
27	Eingang 8
28	Eingang 9
29	Eingang 10
30	Eingang 11
31	Eingang 12
32	Eingang 13
33	Eingang 14
34	Eingang 15
35	Eingang 16
36	GND
37-40	+11,4...32V

8.10 Bestückungspläne

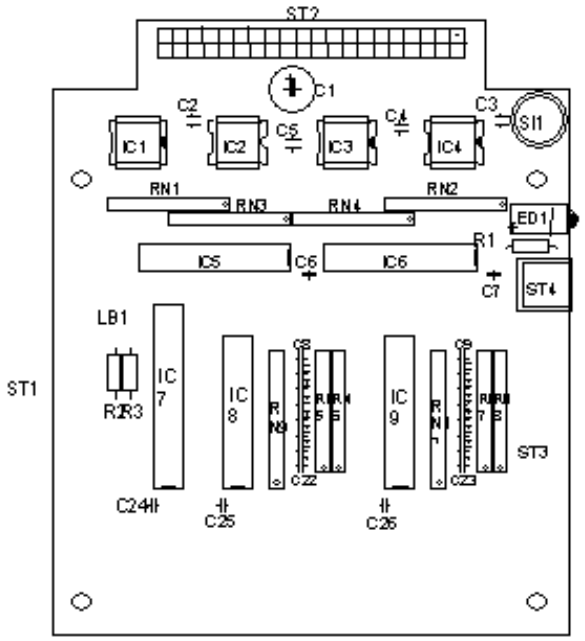
LSTEP PCI-Geberadapterkarte



Die Lotbrücken (LB1-12) befinden sich auf der Lötseite der Platine.

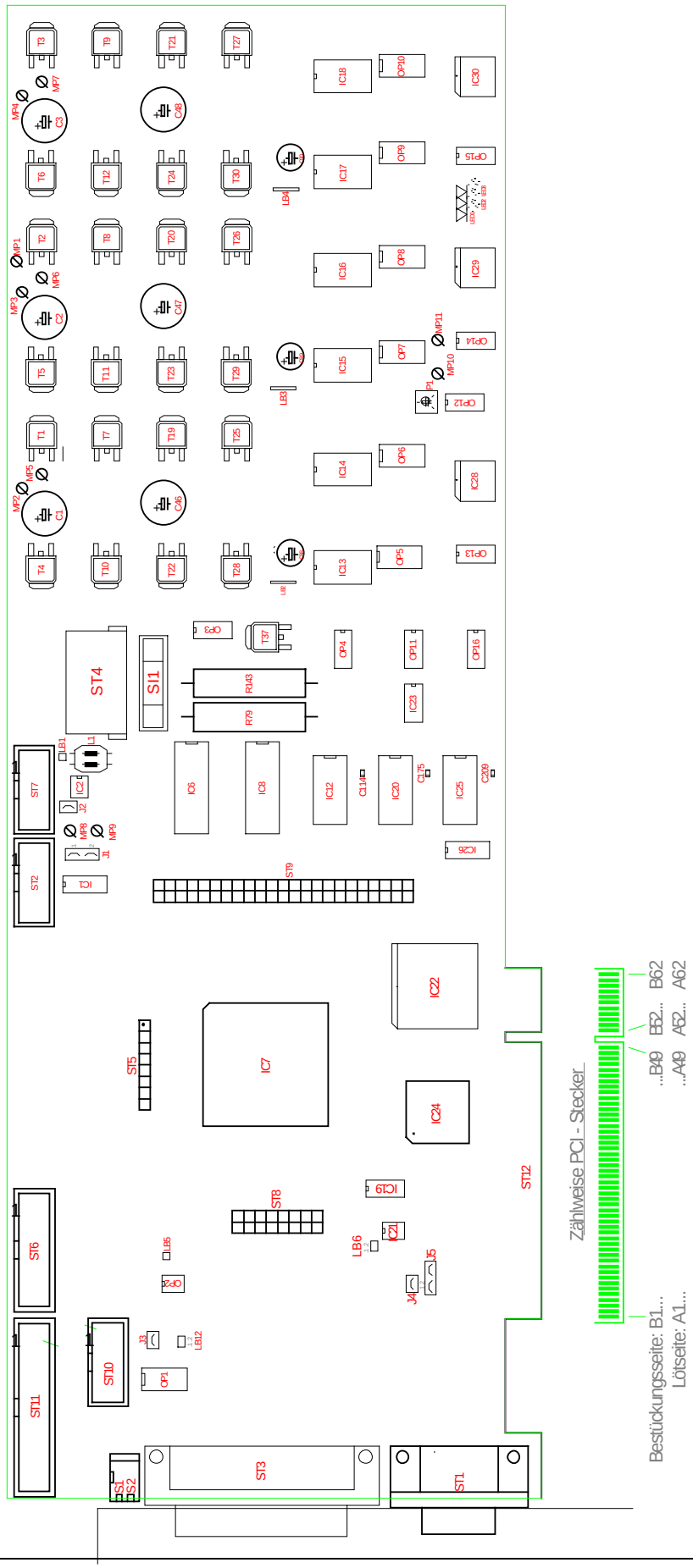


I / O – Adapterkarte für LSTEP PCI 06 07 00

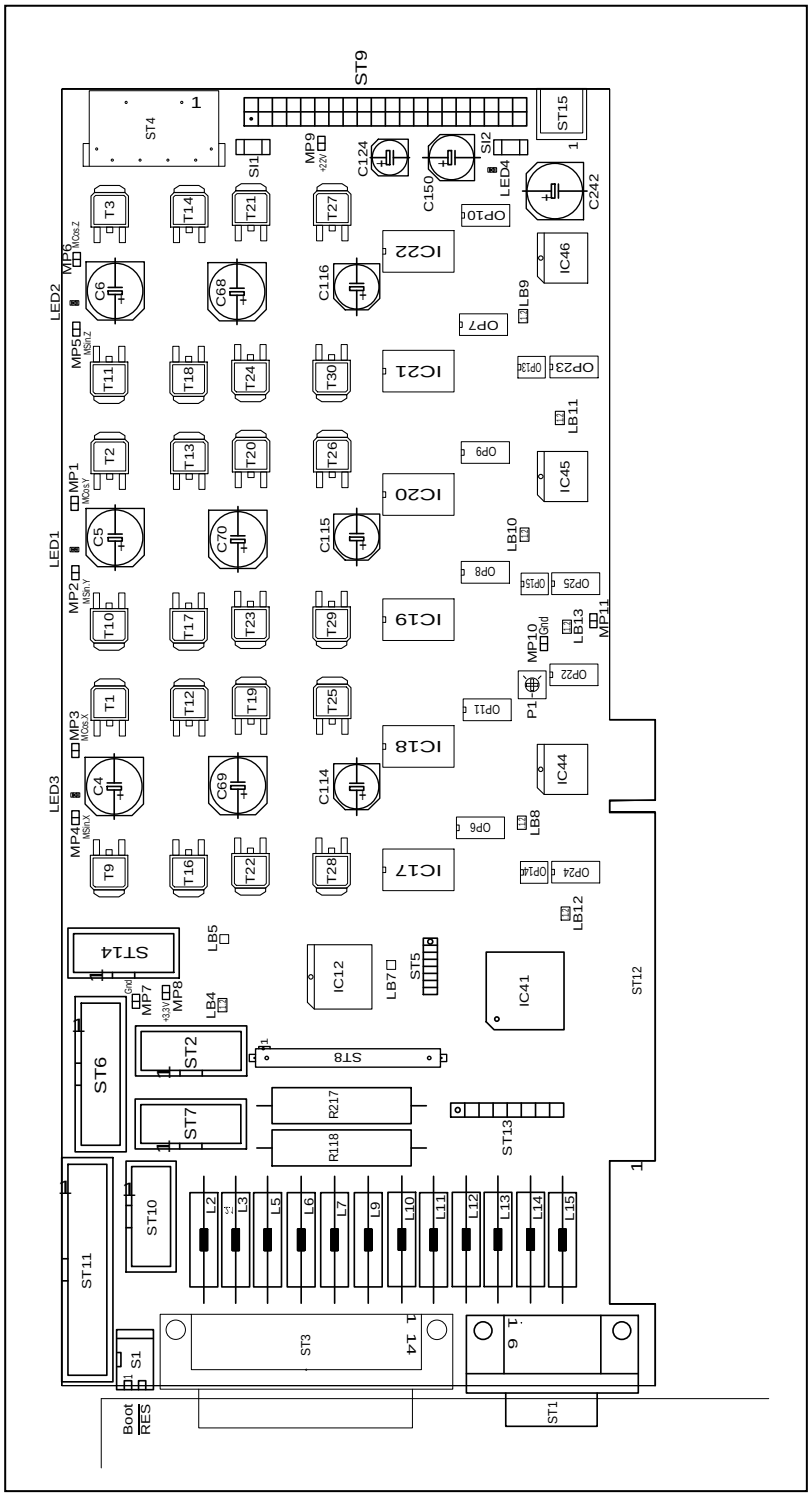


Lötbrücke LB1 befindet sich auf der Platinenlötseite.

LSTEP PCI Bestückungsplan 06 06 00



LSTEP PCIcompact 06 06 07



8.11 Anhang LSTEP-PCI / PCIcompact

Technische Daten

Spannungsversorgung:	Logikspannung über PCI-Slot vom PC Motorspannung: 12V vom PC-Netzteil 11,4-48V über ext. Netzteil		
Max. Motordrehzahl:	40 U/sec. bei 200-schrittigem Motor		
Max. Motorstrom:	1,25A	je Motorphase	LSTEP-PCI / 1
	2,5A	je Motorphase	LSTEP-PCI / 2
	3,75A	je Motorphase	LSTEP-PCI / 3
Max. Motorspannung:	48V		
Schrittauflösung	max. 50.000 Schritte/Umdrehung bei 200schrittigem Motor		
Baudrate:	57,6 Kbd		
Abmessungen L x H x B (1 Slot) PCI PCIcompact	 341mm x 120mm x 20mm (1 Slot) 236mm x 107mm x 15mm (1 Slot)		
max. Zählfrequenz für TTL-Gebereingänge	2,5 Mflanken = 625 KHz -Flankenauswertung		

9 Anhang LSTEP-API

9.1 Einführung

Das LSTEP-API (Programmierschnittstelle für die LStep Feinpositioniersysteme) soll Software-Entwicklern dabei helfen, Anwendungen zur Steuerung der Positioniersysteme LSTEP xx, LSTEP xx/2, LSTEP-PC, ECO-STEP und LSTEP-44 schnell und effektiv zu entwickeln, ohne sich mit hardware-naher Programmierung beschäftigen zu müssen. Es bietet Zugriff auf den kompletten Befehlssatz der LSTEP-Positioniersysteme.

Das LSTEP4X API ist eine um die Unterstützung der parallelen Ansteuerung mehrerer LSTEPs erweiterte Variante des LStep API.

9.1.1 Funktionsumfang

- Windows 32-bit DLL
- Unterstützung der Schrittmotorsteuerungen LSTEP xx, LSTEP xx/2, LSTEP-PC, ECO-STEP, LSTEP-44 und LSTEP-PCI
- Ansteuerung über RS232-Schnittstelle, ISA oder PCI (DPRAM)
- Automatische Erkennung der angeschlossenen Steuerung
- Konfiguration der Steuerung
- Ausführung aller von der Steuerung unterstützten Befehle
- Bis zu 4 Achsen
- Multithreading-fähig

9.1.2 Systemanforderungen

Mit dem LSTEP-API, ebenso wie mit dem LSTEP4X-API können auf Intel-PCs unter MS Windows 9x, Windows NT und Windows 2000 Anwendungen entwickelt werden.

9.1.3 unterstützte Entwicklungsumgebungen

LSTEP-API und LSTEP4X-API wurden mit den folgenden Entwicklungs- und Laufzeitumgebungen getestet:

Borland/Inprise Delphi 3-5

Microsoft Visual C++ 6.0

National Instruments LabVIEW

Sie sollten kompatibel mit allen anderen Programmier-Umgegebungen sein, die DLLs verwenden können.

(DLL = Dynamic Link Library; Eine DLL ist ein ausführbares Modul, das Code und Ressourcen enthält, die von anderen Anwendungen oder DLLs verwendet werden.)

9.2 DLL-Schnittstelle

9.2.1 LSTEP-API

Hauptbestandteil des LSTEP-APIs ist die Datei LSTEP4.DLL. Diese DLL verwenden Sie bei der Entwicklung eigener Programme, um die LSTEP zu konfigurieren, Befehle zu senden, Positionswerte, Ein-/Ausgänge abzufragen etc.

9.2.2 LSTEP4X-API

Hauptbestandteil des LStep4X-APIs ist die Datei LSTEP4X.DLL. Diese DLL verwenden Sie bei der Entwicklung eigener Programme, um mehrere LSTEPs zu konfigurieren, Befehle zu senden, Positionswerte, Ein-/Ausgänge abzufragen etc.

9.2.3 Allgemeine Hinweise

9.2.3.1 LSTEP4.DLL

Die DLL LSTEP4.DLL implementiert die Befehle des LSTEP-API. Alle Funktionen sind mit einem 32-Bit Integer als Rückgabewert deklariert. Eine 0 als Rückgabewert zeigt die fehlerfreie Ausführung der Funktion an, bei Fehlern (z. B. Timeouts) wird der entsprechende Fehlercode (siehe Tabelle) zurückgeliefert.

Bei Funktionen wie LS_MoveAbs werden immer die Werte für 4 Achsen übergeben. Handelt es sich um eine Steuerung mit 1-3 Achsen, werden die Werte für die nicht vorhandenen Achsen ignoriert, sie können auf 0 gesetzt werden.

9.2.3.2 LSTEP4X.DLL

Die DLL LSTEP4X.DLL implementiert die Befehle des LSTEP4X-API. Alle Funktionen sind mit einem 32-Bit Integer als Rückgabewert deklariert. Eine 0 als Rückgabewert zeigt die fehlerfreie Ausführung der Funktion an, bei Fehlern (z. B. Timeouts) wird der entsprechende Fehlercode (siehe Tabelle) zurückgeliefert.

Als erster Parameter wird bei sämtlichen Funktionen des API ein 32-bit Integer-Wert übergeben (zwischen 1 und 32), welcher die Nummer der LStep angibt, an die der Befehl gesendet werden soll.

Die Funktion LSX_CreateLSID kann verwendet werden, um einen solchen ID-Wert zu erzeugen. Mit einem Aufruf von LSX_FreeLSID wird ein ID-Wert wieder freigegeben. (siehe Delphi-Beispiel)

Bei Funktionen wie LSX_MoveAbs werden immer die Werte für 4 Achsen übergeben. Handelt es sich um eine Steuerung mit 1-3 Achsen, werden die Werte für die nicht vorhandenen Achsen ignoriert, sie können auf 0 gesetzt werden.

9.2.3.3 Unterschiede im Vergleich zur LSTEP4.DLL

Am Funktionsumfang hat sich bei der LSTEP4X.DLL im Vergleich zur LSTEP4.DLL nichts geändert, die Funktionsnamen sind identisch. Das normale LStep API (LSTEP4.DLL) wird weitergeführt, bestehender Quellcode, welcher das LStep API nutzt muss also nicht modifiziert werden.

Das LSTEP4X API öffnet für jede LStep ein eigenes Protokollfenster, und die Log-Dateien werden ebenfalls separat für jede LStep geschrieben.

Da das LSTEP4X API Multi-Threading unterstützt, können vom Kunde erstellte Programme von mehreren Threads aus über das API auf die LSteps zugreifen.

Die parallele Ansteuerung mehrerer LSTEP-Schrittmotorsteuerungen ist möglich.

Die Funktionsnamen haben geänderte Prefixe erhalten. Bei der LSTEP4X.DLL werden „LSX_“ anstatt „LS_“ wie bei der LSTEP4.DLL verwendet.

Bei allen Funktionsaufrufen des LSTEP4X API wird als zusätzlicher Parameter ein Integer-Wert übergeben, welcher die Steuerung bezeichnet. Die Numerierung der LSTEPs erfolgt von 1 bis 32.

Hinweis:	Unter Windows NT ist die Installation des mitgelieferten Treibers GIVEIO erforderlich, damit die DPRAM-Schnittstelle der LSTEP-PC verwendet werden können.

9.2.4 Einbindung in Delphi

9.2.4.1 LSTEP4-API

Allen Funktionsnamen des LSTEP4-API ist zur besseren Unterscheidung ein „LS_“ vorangestellt. Um die Funktionen des LSTEP4 APIs verwenden zu können, muss LSTEP4.pas in der uses-Klausel der entsprechenden Unit stehen, und in einem der eingestellten Suchpfade liegen.

benötigte Dateien: LSTEP4.DLL und LSTEP4.pas

Delphi-Beispiel für die Ansteuerung von einer LStep

```
...
var LStep1: Integer;
…
begin

LSX_ConnectSimple(1, 'COM1', 9600, True);

LSX_MoveAbs(10.0, 20.0, 30.0, 0.0, True);

LSX_Disconnect();

end;
```

9.2.4.2

Allen Funktionsnamen des LStep4X-API ist zur besseren Unterscheidung ein „LSX_“ vorangestellt. (siehe LStep4x.pas) Um die Funktionen des LSTEP4X APIs verwenden zu können, muss LSTEP4X.pas in der uses-Klausel der entsprechenden Unit stehen, und in einem der eingestellten Suchpfade liegen.

benötigte Dateien: LSTEP4X.DLL und LSTEP4X.pas

Delphi-Beispiel für die parallele Ansteuerung von 2 LSteps

```
...
```

```
var LStep1, LStep2: Integer;
…
begin
LSX_CreateLSID(LStep1);
LSX_CreateLSID(LStep2);


LSX_ConnectSimple(LStep1, 1, 'COM1', 9600, True);
LSX_ConnectSimple(LStep2, 1, 'COM2', 9600, True);

LSX_MoveAbs(LStep1, 10.0, 20.0, 30.0, 0.0, True);
LSX_MoveAbs(LStep2, 5.0, 10.0, 0.0, 0.0, True);


LSX_Disconnect(LStep1);
LSX_Disconnect(LStep2);

LSX_FreeLSID(LStep1);
LSX_FreeLSID(LStep2);

end;
```

9.2.5 Einbindung in Visual C++

9.2.5.1 LSTEP4-API

Für Visual C++ wurde eine Kapselung der LSTEP4.DLL erstellt. Die Klasse CLStep4 lädt die DLL und alle Zeiger auf Funktionsaufrufe dynamisch. Den Methoden des LSTEP-Objekts ist kein „LS_“ vorangestellt.
(Beispiel: LS.Calibrate() statt LS_Calibrate)
Von der Klasse CLStep4 sollte nur eine Instanz erstellt werden, zumal mit dem LSTEP API momentan nicht mehrere LSteps gleichzeitig gesteuert werden können.

benötigte Dateien: LSTEP4.DLL, LSTEP4.h und LSTEP4.cpp

Visual C++-Beispiel für die Ansteuerung von einer LStep

```
...
CLStep4 LS1;
…

LS1.ConnectSimple(1, "COM1", 9600, true);

LS1.MoveAbs(10.0, 20.0, 30.0, 0.0, true);

LS1.Disconnect();
delete LS1;
```

9.2.5.2 LSTEP4X-API

Für Visual C++ wurde eine Kapselung der LSTEP4X.DLL erstellt. Die Klasse CLStep4X lädt die DLL und alle Zeiger auf Funktionsaufrufe dynamisch. Den Methoden des LSTEP Objekts ist kein „LSX_“ vorangestellt.
(Beispiel: LSX.Calibrate() statt LSX_Calibrate)

Die Funktionen LSX_CreateLSID und LSX_FreeLSID müssen Sie in C++ zur Verwendung der LSTEP4X.DLL nicht aufrufen, da die Wrapper-Klasse CLStep4X den Integer-Wert, welcher die Nummer der LStep angibt, selbst verwaltet. Die Methoden von CLStep4X besitzen also keinen zusätzlichen Parameter für die Nummer der LStep.

benötigte Dateien: LSTEP4X.DLL, LSTEP4X.h und LSTEP4X.cpp

Visual C++-Beispiel für die parallele Ansteuerung von 2 LSteps

```
...
CLStep4X* LS1,* LS2;
…

LS1 = new CLStep4X;
LS2 = new CLStep4X;

LS1->ConnectSimple(1, "COM1", 9600, true);
LS2->ConnectSimple(1, "COM2", 9600, true);

LS1->MoveAbs(10.0, 20.0, 30.0, 0.0, true);
LS2->MoveAbs(5.0, 10.0, 0.0, 0.0, true);

LS1->Disconnect();
delete LS1;
LS2->Disconnect();
delete LS2;
```

9.2.6 Einbindung in LabVIEW

NI LabVIEW ist eine auf der graphischen Programmiersprache G basierende Entwicklungsumgebung. Sie ermöglicht eine vereinfachte und schnelle Programmierung mit graphischen Symbolen. Die Erstellung komplilierter 32-bit Programme ist möglich, sodaß die nötige Ausführungsgeschwindigkeit für Steuerungs-, Test- und Meßanwendungen gegeben ist.

Alle LabVIEW-Programme (sogenannte VIs, Virtual Instruments) besitzen ein Frontpanel und ein Blockdiagramm, und können wiederum als Unterprogramm (SubVI) in andere Programme eingebunden werden.

Für die Einbindung der LSTEP-API (LSTEP4.DLL und LSTEP4X.DLL) wurden VI-Bibliotheken (LSTEP4.LLB bzw LSTEP4X.LLB) erstellt, die jeweils ca. 110 VIs enthalten. Diese einzelnen VIs (z.B. LS4 ConnectSimple.vi) kapseln die entsprechenden LSTEP API-Funktionen. Die LSTEP4.DLL bzw. LSTEP4X.DLL wird mittels ‚Call Library Function‘ (Aufruf ext. Bibliotheken) verwendet.

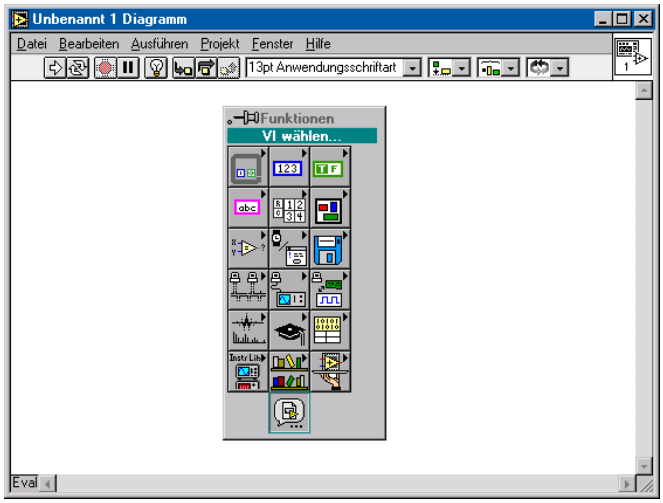
9.2.6.1 Unterschiede zwischen LSTEP4.LLB und LSTEP4X.LLB

Bei neuen Software-Projekten mit dem LSTEP-API unter LabView sollten Sie ausschließlich die VI-Bibliothek LSTEP4X.LLB verwenden. Dies hat folgenden Grund: Die LSTEP4.LLB unterstützt nur maximale eine LStep, ausserdem haben hier alle VI's als Rückgabewert nur eine Boolesche Variable, also keinen interpretierbaren Fehlercode.

In der (neueren) LSTEP4X.LLB haben nun die VI's als Rückgabewert einen 32-bit-Integer-Wert (Dieser ist als „error out" bezeichnet). Ist dieser gleich 0, so signalisiert dies daß die LSTEP-API-Funktion fehlerfrei ausgeführt wurde.Anderenfalls kann die Bedeutung des Fehlercodes der Tabelle in dieser Dokumentation entnommen werden. Ausserdem können durch die in den VI's aufgerufene LSTEP4X.DLL mehrere Lsteps parallel angesteuert werden. Die VI's für die LSTEP4X.DLL unterscheiden sich im Dateinamen von denen der LSTEP4.DLL dadurch, daß sie mit dem Kürzel „LS4X" statt „LS4" beginnen. In LabView haben die VI's für die LSTEP4X.DLL die Hintergrundfarbe lila, die der LSTEP4.DLL die Hintergrundfarbe blau. Die Benennung der VI's in den Symbolen ist dieselbe. Bei allen VI's kommt ein zusätzlicher Anschluß hinzu. Dieser ist ein 32-bit-Integer-Wert und gibt die Nummer der LStep an, auf die sich der Befehl, z.B. ein Verfahrbefehl oder das Auslesen der aktuellen Position, bezieht („LStep Controller ID"). Diese Nummern können Sie entweder selbst vergeben (z.B. „0" für die LStep an der seriellen Schnittstelle COM1, „1" für die LStep an COM2 etc.), oder über das VI „LS4X CreateLSID" erzeugen lassen. Mit dem VI „LS4X FreeLSID" können erzeugte LStep ID-Nummern wieder „freigegeben" werden. Wenn Sie in Ihrem LabView-Projekt nur eine LStep verwenden, können Sie den Anschluß „LStep Controller ID" bei allen verwendeten VI's aus LSTEP4X.LLB freilassen, denn dieser hat einheitlich den Default-Wert 1.benötigte Dateien in LabView: LSTEP4X.DLL und LSTEP4X.LLB
bzw. LSTEP4.DLL und LSTEP4.LLB

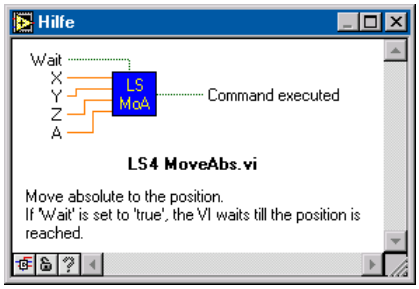
9.2.6.2 Vorgehensweise zur Verwendung eines LSTEP4-VI's

1. Neues VI erstellen
2. Zu Blockdiagramm-Fenster wechseln (Strg+E)
3. Auf Diagramm klicken (rechte Maustaste)

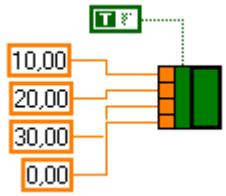


4. VI wählen...
5. Im Dateidialog die mitgelieferte VI-Bibliothek LSTEP4.LLB öffnen, daraus dann den gewünschten Befehl wählen (z.B. LS4 MoveAbs.vi)
6. VI im Diagramm plazieren

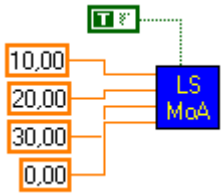
Die Tastenkombination Strg+H öffnet ein Hilfe-Fenster, dieses zeigt Informationen über das VI an, auf dem sich momentan der Mauszeiger befindet



Die Übergabe von Parametern an SubVIs erfolgt über Anschlüsse, die ‚verkabelt' werden müssen. Damit diese Anschlüsse im Diagramm gezeigt werden, klicken Sie mit der rechten Maustaste auf das VI und wählen ‚Anzeigen/Anschlüsse'. Anschließend können Sie den Anschlüssen Werte/Quellen zuweisen. Einer von mehreren möglichen Wegen: Klicken Sie mit der rechten Maustaste auf den gewünschten Anschluss, dann auf den Menüpunkt ‚Konstante erzeugen'.



In diesem Beispiel wird ein absoluter Verfahrbefehl (X 10mm, Y 20mm, Z 30mm) ausgeführt.

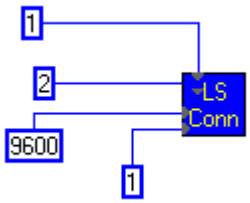


Die Belegung der Anschlüsse der VIs kann der Dokumentation der entsprechenden API-Funktion entnommen werden, dort findet sich die graphische Darstellung, welche auch im Hilfe-Fenster von LabVIEW erscheint. Die Parameter entsprechen weitgehend denen der DLL-Funktion: Lediglich bei Funktionen, denen Bitmasken als Parameter übergeben werden, sind einige Unterschiede vorhanden (z.B. LS4 SetActiveAxes.vi)

Die LSTEP4 VIs verfügen über einen Anschluss namens ‚Command executed'. Ist dieser boolesche Wert ‚true', wurde der Befehl erfolgreich ausgeführt. Falls ein Fehler aufgetreten ist, wird der Wert auf ‚false' gesetzt.

Bevor Verfahrbefehle ausgeführt, Positionswerte ausgelesen werden können etc. muß die Verbindung zur LSTEP geöffnet werden. Dies ist am einfachsten über das VI ‚LS4 ConnectSimple.vi' möglich. Es initialisiert die Schnittstelle und erkennt die angeschlossene LSTEP.

Beispiel für RS232 (COM2 und 9600 Baud):

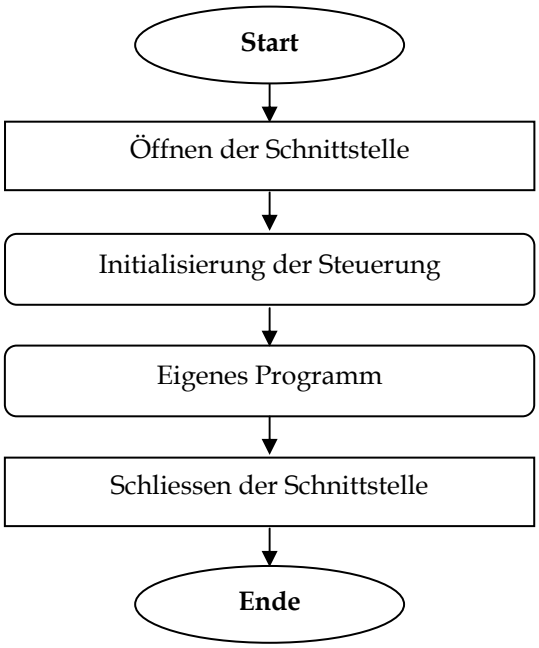


9.3 Hinweise zum Aufbau eigener Programme bei der Programmierung der Steuerungen über das API

Die folgenden Abbildung zeigt den Programmflussplan nach dem die Programme zur Steuerung Positioniersysteme aufgebaut sein sollten. Die verwendeten Funktionen sind in der Beschreibung des LSTEP-API aufgelistet und werden dort genauer beschrieben.

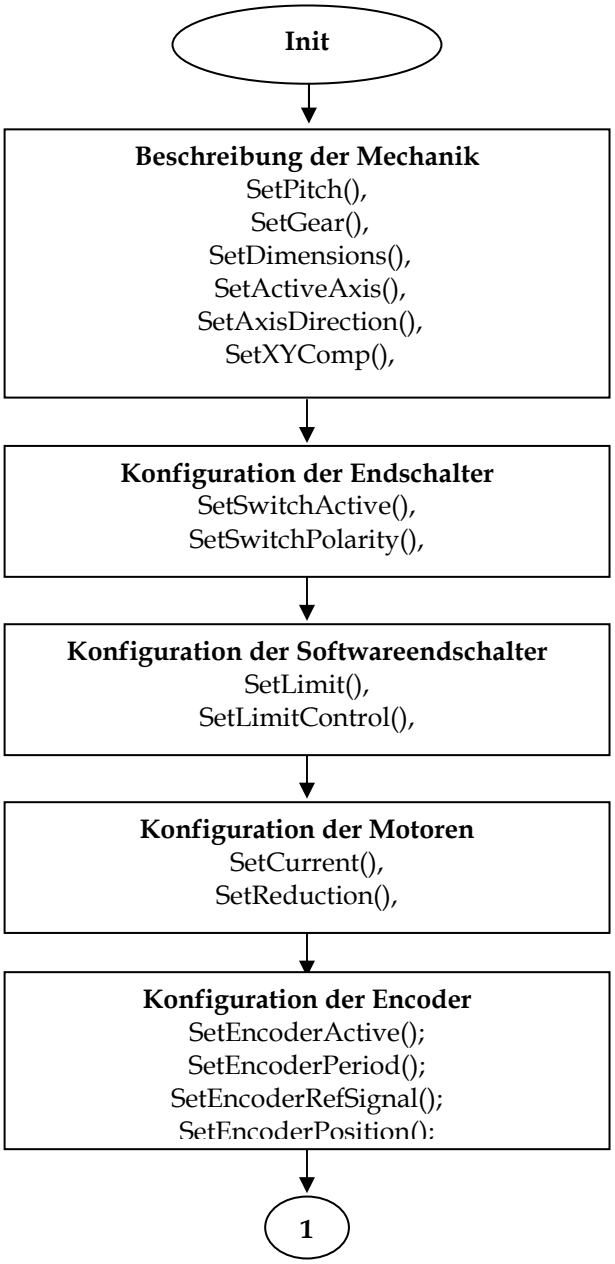
Da die vorkonfigurierten Default-Settings der Steuerung nicht für alle Anwendungen die entsprechenden Daten enthalten können sind, nachdem die verwendete Schnittstelle geöffnet worden ist, die im Punkt „Initialisierung der Steuerung“ beschriebenen Schritte vorzunehmen.

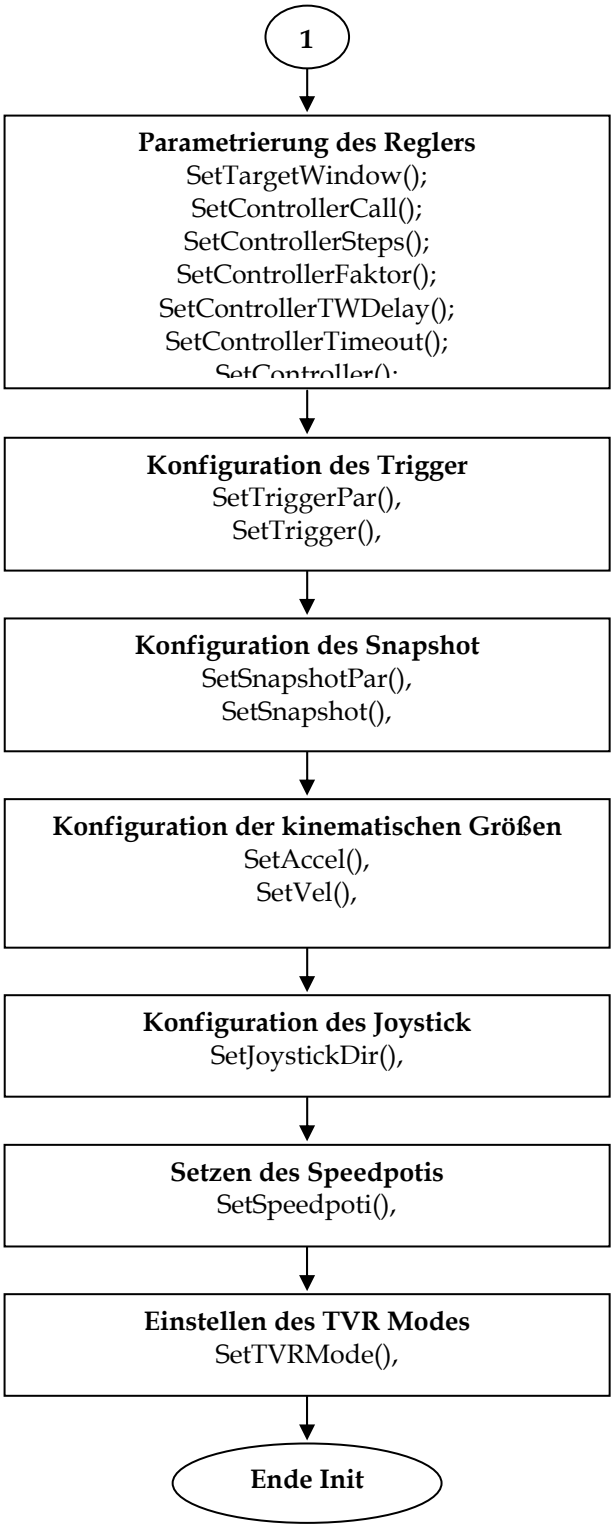
Anschliessend kann unter Verwendung des LSTEP-API ein beliebiges Anwenderprogramm geschrieben werden.



9.3.1 Initialisierung der Steuerung

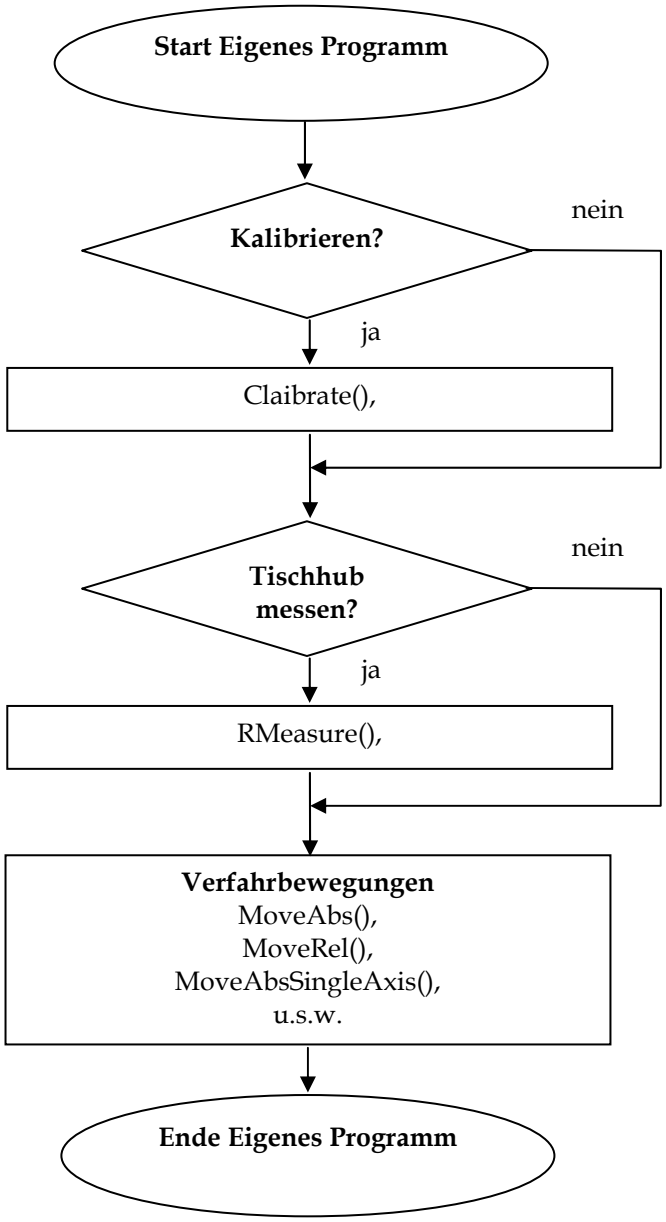
Vor dem Start des eigenen Programmteils sind bei der Initialisirung der verwendeten LSTEP die im folgenden beschriebenen Grundeinstellungen der Positioniersteuerung vorzunehmen um einen fehlerfreien Betrieb zu gewährleisten.




1
Parametrierung des Reglers
SetTargetWindow();
SetControllerCall();
SetControllerSteps();
SetControllerFaktor();
SetControllerTWDelay();
SetControllerTimeout();
SetController();
Konfiguration des Trigger
SetTriggerPar(),
SetTrigger(),
Konfiguration des Snapshot
SetSnapshotPar(),
SetSnapshot(),
Konfiguration der kinematischen Größen
SetAccel(),
SetVel(),
Konfiguration des Joystick
SetJoystickDir(),
Setzen des Speedpotis
SetSpeedpoti(),
Einstellen des TVR Modes
SetTVRMode(),
Ende Init

9.3.2 Eigener Programmteil

Im eigenen Programmteil kann der Anwender die gewünschte Funktionalität der Steuerung programmieren. Dazu zählen das Ausführen von Positionierbewegungen in Abhängigkeit der Zustände von digitalen I/Os ebenso wie das Setzen von Triggersignalen in Abhängigkeit von der Position usw..



9.4 Funktionen

9.4.1 Index für API-Befehle

Gliederung der Befehle nach Themen wie folgt:

API-Konfiguration/Schnittstelle

Steuerungs-Info:

Einstellungen

SetMotorTablePatch	Korrekturtabelle Ein/Aus	49
SetOutFuncLev	Setzt die StromUmschaltungsgeschwindigkeit	50
SetPitch	Spindelsteigung setzen	51
SetPowerAmplifier	Schaltet bei LS44 Endstufen Ein/Aus	52
SetReduction	Stromabsenkung einstellen	53
SetRefSpeed	Geschwindigkeit, mit der beim Kalibrieren die Referenzmarke gesucht wird, setzen	54
SetRMOffset	RM-Offset	55
SetSpeedPoti	Speed-Poti Ein/ Aus	56
SetStopAccel	Stellt die Bremsbeschleunigung ein, wenn der Stopeingang aktiv ist	57
SetStopPolarity	Stopeingang Polarität einstellen	58
SetVel	Geschwindigkeit aller Achsen einstellen	59
SetVelFac	Geschwindigkeitsuntersetzung setzen	60
SetVelSingleAxis	Geschwindigkeit für eine Achse einstellen	64
SetVLevel	Ausblenden von Geschwindigkeiten, bei denen Resonanzen auftreten	62
SetXYAxisComp	XY-Achsüberlagerung aktivieren	63
SoftwareReset	Software wird in den Startzustand versetzt	64

Statusabfragen

Befehl	Kurzbeschreibung	Seite
GetError	liefert die aktuelle Fehlernummer	65
GetSecurityErr	Liest alle Zustände und Ergebnisse der GAL-Sicherheitsüberwachung (nur bei LS44-Steuerungen)	66
GetSecurityStatus	Liest den aktuellen Zustand der Sicherheitsüberwachung	67
GetStatus	liefert den aktuellen Zustand der Steuerung	68
GetStatusAxis	liefert den aktuellen Zustand der einzelnen Achsen	68
GetStatusLimit	Liefert den aktuellen Zustand der Software-Grenzen jeder einzelnen Achse	69
SetAutoStatus	AutoStatus Ein/Aus	69

Fahrbefehle und Positionsverwaltung

Befehl	Kurzbeschreibung	70
Calibrate	Kalibrieren	70
CalibrateEx	Es werden nur die Achsen kalibriert, deren entsprechendes Bit in dem übergebenen Integer-Wert gesetzt ist.	70
Clearpos	Positionswerte werden genullt (für endlos Drehachsen)	71
GetDelay	Liefert die Verzögerung des Vektorstarts	71
GetDistance	Liefert die Strecke, die mit LS_PosRelShort gestartet wird	72
GetPos	Abfrage der aktuellen Position aller Achsen	73
GetPosEx	Abfrage der aktuellen Geber- bzw. Positionswerte aller Achsen	73
GetPosSingleAxis	Abfrage der aktuellen Position einer Achse	74
MoveAbs	Absolutposition anfahren	75
MoveAbsSingleAxis	Absolutposition einer Achse anfahren	75
MoveEx	Erweiterter Verfahr-Befehl	76
MoveRel	Relativen Vektor fahren	77
MoveRelShort	Positionieren Relativ (short command)	77
MoveRelSingleAxis	Relativen Vektor einer Achse verfahren	78
RMeasure	Tischhub messen	78
RmeasureEx	Tischhub messen wird nur bei den Achsen durchgeführt, deren	79

	entsprechendes Bit in dem übergebenen Integer-Wert gesetzt ist	
SetDelay	Durch den Befehl Delay kann eine Verzögerung des Vektorstarts erzeugt werden	71
SetDistance	Strecke setzen (für MoveRelShort)	72
SetPos	Position setzen	79
StopAxes	alle Verfahrbewegungen werden abgebrochen	80
WaitForAxisStop	Die Funktion kehrt zurück, sobald die in der Bit-Maske AFlags gewählten Achsen ihre Zielposition erreicht haben	80

Joystick und Handrad

Befehl	Kurzbeschreibung	Seite
GetDigJoySpeed	Digitaler Joystick und Geschwindigkeit lesen	81
GetHandwheel	Liest den Zustand des Handrads	82
GetJoystick	Liest den Zustand des Analog-Joysticks	82
GetJoystickDir	Richtung Joystick	83
GetJoystickWindow	Joystick-Fenster ablesen	84
SetDigJoySpeed	Digitaler Joystick und Geschwindigkeit setzen	81
SetDigJoyOff	Schaltet digitalen Joystick aus	85
SetHandwheelOff	Handrad Aus	86
SetHandwheelOn	Handrad Ein	86
SetJoystickDir	Richtung Joystick	83
SetJoystickOff	Analog-Joystick Aus	87
SetJoystickOn	Analog-Joystick Ein	87
SetJoystickWindow	Joystick-Fenster setzen	84
GetJoyChangeAxis	Liest Joystickachszuordnung	84
JoyChangeAxis	Setzt Joystickachszuordnung	85

Bedienpult mit Trackball und Joyspeed-Tasten

Befehl	Kurzbeschreibung	Seite
GetBPZ	Liest Zustand des Bedienpults	88
GetBPZJoyspeed	Liest Bedienpult Joystick-Speed	89
GetBPZTrackballBackLash	Liest Bedienpult Trackball-Umkehrspiel	90
GetBPZTrackballFactor	Liest Bedienpult Trackball-Faktor	91
SetBPZ	Bedienpult Ein/Aus	88
SetBPZJoyspeed	Bedienpult Joystick-Speed	89
SetBPZTrackballBackLash	Bedienpult Trackball-Umkehrspiel	90
SetBPZTrackballFactor	Bedienpult Trackball-Faktor	91

Endschalter (Hardware u. Software)

Befehl	Kurzbeschreibung	Seite
GetAutoLimitAfterCalibRM	Gibt an, ob beim Kalibrieren und Tischhubmessendie interne Software-Limits gesetzt werden.	92
GetLimit	Liefert Verfahrbereichsgrenzen	93
GetLimitControl	Liest,ob die Bereichsüberwachung eingeschaltet ist	94
GetSwitchActive	Gibt an, ob die Endschalter eingeschaltet sind	95
GetSwitches	liest den Zustand aller Endschalter	96
GetSwitchPolarity	Liest Endschalterpolarität	97
SetAutoLimitAfterCalibRM	Verhindert, dass beim Kalibrieren und Tischhubmessen die internen Software-Limits gesetzt werden.	92
SetLimit	Verfahrbereichsgrenzen einstellen	93
SetLimitControl	Bereichsüberwachung	94
SetSwitchActive	Liest Status für Endschalter Ein / Aus	95
SetSwitchPolarity	Endschalterpolarität einstellen	97

Digitale und analoge Ein.- und Ausgänge

Befehl	Kurzbeschreibung	Seite
GetAnalogInput	Lesen des aktuellen Zustands eines Analogkanals	98
GetAnalogInputs2	Lesen der aktuellen Zustände der Analogkanäle PT100, MV und V24	98
GetDigitalInputs	Alle Inputpins lesen	99
GetDigitalInputsE	Zusätzliche digitale Eingänge lesen (16-31)	99
SetAnalogOutput	Analogkanal setzen	99
SetDigIO_Distance	Aktivierung eines Ausgang in Abhängigkeit der eingestellten Strecke vor/nach der Zielposition	100
SetDigIO_EmergencyStop	Funktion der digitalen Ein-/Ausgänge Zuordnung des Not-Stop-Pins	100
SetDigIO_Off	Funktion der digitalen Ein-/Ausgänge Aus	101
SetDigIO_Polarity	Einstellung der Polarität	101
SetDigitalOutput	Digitalen Ausgang setzen	102
SetDigitalOutputs	Digitale Ausgänge setzen (0-15)	102
SetDigitalOutputsE	Zusätzliche digitale Ausgänge setzen (16-31)	102

Takt-Vor/Rück Eingänge

Befehl	Kurzbeschreibung	Seite
GetFactorTVR	Liest den Faktor Takt Vor / Rück	103
GetTVRMode	Einstellung vom Takt Vor / Rück auslesen	104
SetFactorTVR	Faktor Takt Vor / Rück	103
SetTVRMode	Takt Vor / Rück einstellen	104

Takt-Vor/Rück über Schnittstelle

Befehl	Kurzbeschreibung	Seite
SetTVRInPulse	Takt-Vor/Rück über Schnittstelle	105

Takt-Vor/Rück Ausgänge für weitere Achsen

Befehl	Kurzbeschreibung	Seite
GetAccelTVRO	Alle eingestellten Beschleunigungen lesen	106
GetPosTVRO	Liefert Positionswerte, in Abhängigkeit von Dimension	107
GetStatusTVRO	Liefert den aktuellen Status der Achsen	108
GetTVROutMode	Einstellung vom Takt Vor/Rück lesen	109
GetTVROutPitch	Liest Spindelsteigung	110
GetTVROutResolution	Liefert dieAuflösung der an zu steuernden Endstufe	111
GetVelTVRO	Alle eingestellten Geschwindigkeiten lesen	112
MoveAbsTVROSingleAxis	Absolutposition einer Achse anfahren	113
MoveAbsTVRO	Absolutposition anfahren	113
MoveRelTVROSingleAxis	Relativen Vector einer Achse verfahren	114
MoveRelTVRO	Relativen Vector verfahren	114
SetAccelSingleAxisTVRO	Beschleunigung einer einzelnen Achse setzen	115
SetAccelTVRO	Beschleunigungen setzen	106
SetPosTVRO	Position setzen	107
SetTVROutMode	Takt Vor / Rück einstellen	109
SetTVROutPitch	Spindelsteigung setzen	110
SetTVROutResolution	Auflösung der an zu steuernden Endstufe	111
SetVelSingleAxisTVRO	Geschwindigkeit einer einzelnen Achse setzen	115
SetVelTVRO	Geschwindigkeiten setzen	112

Geber-Einstellungen

Befehl	Kurzbeschreibung	Seite
ClearEncoder	Geberposition auf null setzen	116
GetEncoder	Liest alle Geberpositionen	116
GetEncoderActive	Liest, welche Geber nach der Kalibration aktiv werden	117
GetEncoderMask	Geberzustände auslesen	118
GetEncoderPeriod	Geberperiodenlängen auslesen	119
GetEncoderPosition	Liefert Einstellung von Geberwertanzeige	120
GetEncoderRefSignal	Gibt an, ob beim Kalibrieren Referenzsignal von Geber ausgewertet werden soll	121
SetEncoderActive	Mit dieser Funktion kann ausgewählt werden, welche Geber nach der Kalibration aktiviert werden sollen	117
SetEncoderMask	Geber (de-)aktivieren	118
SetEncoderPeriod	Geberperiodenlängen einstellen	119
SetEncoderPosition	Geberwertanzeige Ein/Aus	120
SetEncoderRefSignal	Beim Kalibrieren Referenzsignal von Geber auswerten	121

Reglereinstellungen

Befehl	Kurzbeschreibung	Seite
ClearCtrFastMoveCounter	Anzahl ausgeführter FastMove Funktionen auf 0 setzen	122
GetController	Regler-Modus auslesen	123
GetControllerCall	Einstellung vom Regleraufruf auslesen	124
GetControllerFactor	Einstellung vom Reglerfaktor auslesen	125
GetControllerSteps	Regler-Schritte auslesen	126
GetControllerTimeout	Liefert die Einstellung vom Regler-Überwachungs-Timeout	127
GetControllerTWDelay	Reglerverzögerung auslesen	128
GetCtrFastMove	Einstellung von Fast Move Funktion lesen	129
GetCtrFastMoveCounter	Anzahl ausgeführter FastMove Funktionen auf 0 auslesen	129
GetTargetWindow	Liefert Reglerzielfenster	130
SetController	Regler-Modus einstellen	123
SetControllerCall	Regleraufruf einstellen	124
SetControllerFactor	Reglerfaktor einstellen	125
SetControllerSteps	Regler-Schritte einstellen	126
SetControllerTimeout	Regler-Überwachungs-Timeout einstellen [ms]	127
SetControllerTWDelay	Reglerverzögerung setzen	128
SetCtrFastMoveOff	Fast Move Funktion „AUS“	131
SetCtrFastMoveOn	Fast Move Funktion „EIN“	131
SetTargetWindow	Reglerzielfenster einstellen	130

Trigger-Ausgang

Befehl	Kurzbeschreibung	Seite
GetTrigCount	Triggerzählerstand setzen	132
GetTrigger	Einstellung vom Trigger auslesen	133
GetTriggerPar	Trigger Parameter auslesen	134
SetTrigCount	Triggerzählerstand lesen	132
SetTrigger	Trigger Ein/ Aus	133
SetTriggerPar	Trigger Parameter	134

Snapshot-Eingang

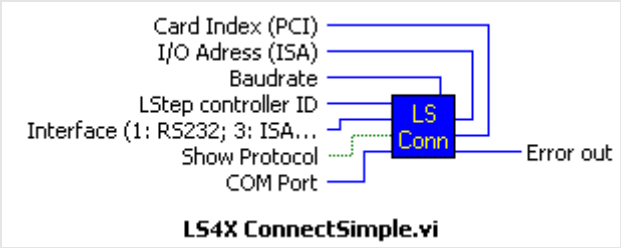
Befehl	Kurzbeschreibung	Seite
GetSnapshot	Einstellung vom Snapshot auslesen	135
GetSnapshotCount	Snapshot-Zähler	136
GetSnapshotFilter	Eingangsfilter auslesen	136
GetSnapshotPar	Snapshot-Parameter auslesen	137
GetSnapshotPos	Snapshot-Position auslesen	138
GetSnapshotPosArray	Snapshot-Position aus Array auslesen	138
SetSnapshot	Snapshot Ein/Aus	135
SetSnapshotFilter	Eingangsfilter setzen	136
SetSnapshotPar	Snapshot-Parameter	137

9.4.2 Funktionen

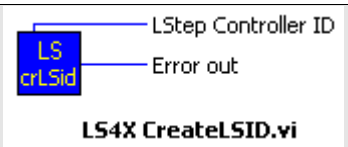
API-Konfiguration/Schnittstelle

LS_Connect	
Beschreibung:	Mit LSTEP verbinden
	Dazu werden die Schnittstellen-Parameter verwendet, welche mittels LS_LoadConfig aus der INI-Datei geladen wurden. (Eine der Funktionen LS_Connect, LS_ConnectSimple oder LS_ConnectEx muß zur Initialisierung der Schnittstelle aufgerufen werden, damit die Kommunikation mit der LSTEP möglich ist.)
Delphi:	function LS_Connect: Integer; function LSX_Connect(LSID: Integer): Integer;
C++:	int Connect();
LabView:	
Parameter:	-
Beispiel:	LS.LoadConfig("C:\LStepTest\LStep.INI"); LS.Connect();

LS_ConnectEx	
Beschreibung:	Mit LSTEP verbinden
	Diese Funktion bietet erweitere Möglichkeiten, übergeben wird ein Zeiger auf eine Datenstruktur, die die Schnittstellenparameter enthält. In dem Record werden außerdem Informationen über die erkannte Steuerung zurückgeliefert (Versionsnummer...) (Eine der Funktionen LS_Connect, LS_ConnectSimple oder LS_ConnectEx muß zur Initialisierung der Schnittstelle aufgerufen werden, damit die Kommunikation mit der LSTEP möglich ist.)
Delphi:	function LS_ConnectEx(var AControlInitPar: TLS_ControlInitPar): Integer; function LSX_ConnectEx(LSID: Integer; var AControlInitPar: TLS_ControlInitPar): Integer;
C++:	int ConnectEx (TLS_ControlInitPar *pAControlInitPar);
Parameter:	AControlInitPar: Zeiger auf einen Record des Typs TLS_ControlInitPar
Beispiel:	LS.ConnectEx(&ControlInitPar1);

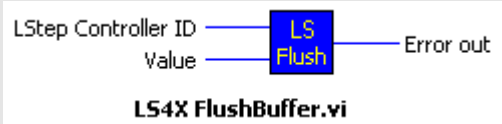
LS_ConnectSimple	
Beschreibung:	Mit LSTEP verbinden
	Die Einstellungen der Schnittstelle werden als Parameter übergeben (Eine der Funktionen LS_Connect, LS_ConnectSimple oder LS_ConnectEx muß zur Initialisierung der Schnittstelle aufgerufen werden, damit die Kommunikation mit der LSTEP möglich ist.)
Delphi:	function LS_ConnectSimple(AnInterfaceType: Integer; AComName: PChar; ABR: Integer; AShowProt: LongBool): Integer; function LSX_ConnectSimple(LSID: Integer; AnInterfaceType: Integer; AComName: PChar; ABaudRate: Integer; AShowProt: LongBool): Integer;
C++:	int Connect (int lAnInterfaceType, char *pcAComName, int lABR, BOOL AShowProt);
LabView:	
Parameter:	AnInterfaceType: Schnittstellentyp 1 = RS232 2 = ArcNet 3 = DPRAM / ISA-Bus 4 = DPRAM / PCI-Bus 11= RS232 mit RTS/CTS Auswertung AComName: Name der COM-Schnittstelle, z. B. 'COM2', bei ArcNet oder DPRAM auf NULL setzen ABR: Bedeutung ist abhängig vom Schnittstellentyp RS232 → Baudrate, z. B. 9600 ArcNet: → 0 für Koax, 1 für Twisted Pair DPRAM / ISA-Bus: → Basis-I/O-Adresse der Karte, z.B. 0x0340 DPRAM / PCI-Bus: → O=erste Karte 1=zweite Karte AShowProt: bestimmt, ob das Schnittstellenprotokoll angezeigt werden soll

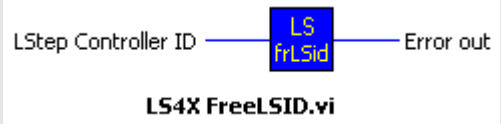
Beispiel:	LS.ConnectSimple(1, "COM2", 9600, true);
	// RS232, 9600 Baud
	oder
	LS_ConnectSimple(4, nil, 0, true);
	//LStep PCI Karte 0;

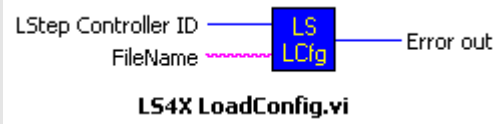
LSX_CreateLSID (nur LSTEP4X-API)	
Beschreibung	Erzeugt eine LStep-ID-Nummer. Diese wird als zusätzlicher Parameter bei den LSTEP4X-API- Befehlen verwendet, um aus mehreren angeschlossenen LSteps die LStep zu wählen, auf die sich der Befehl beziehen soll.
Delphi	function LSX_CreateLSID(var LSID: Integer): Integer;
C++	-
LabView:	
Parameter	LSID: enthält nach Aufruf von CreateLSID eine neue LStep-ID-Nummer, die dann für Connect-, Verfahrbefehle und andere Befehle verwendet werden kann
Beispiel	var LStep1: Integer; ... LSX_CreateLSID(&LStep1);

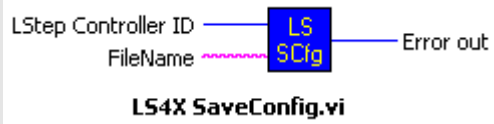
LS_Disconnect	
Beschreibung:	Verbindung zu LSTEP trennen
	Nach Aufruf dieser Funktion können keine Befehle mehr zur LSTEP gesendet werden. Die Funktion sollte kurz vor Beendigung des Programms aufgerufen werden.
Delphi:	function LS_Disconnect: Integer; function LSX_Disconnect(LSID: Integer): Integer;
C++:	int Disconnect ();
LabView:	
Parameter:	-
Beispiel:	LS.Disconnect();

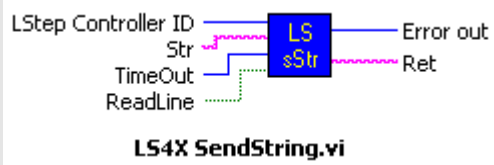
LS_EnableCommandRetry	
Beschreibung	Mit dieser Funktion kann das wiederholte Senden von Kommandos im Falle von Fehlern ein-/ausgeschaltet werden (Standardmäßig ist dieses eingeschaltet)
Delphi	function LS_EnableCommandRetry(AValue: LongBool): Integer; function LSX_EnableCommandRetry(LSID: Integer; AValue: LongBool): Integer;
C++	int EnableCommandRetry (BOOL bAValue);
LabView:	
Parameter	AValue: true => bei Fehlern wiederholt das LStep API das Senden bestimmter Kommandos (insbesondere bei WaitForAxisStop) false => wiederholtes Senden abschalten
Beispiel	LS.EnableCommandRetry(false) ;

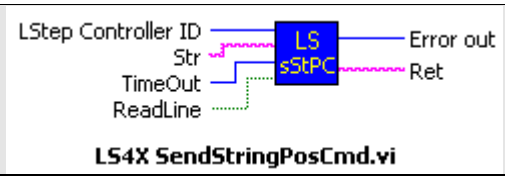
LS_FlushBuffer	
Beschreibung:	Kommunikations-Eingabepuffer löschen (RS-232 und PCI) kann in Fehler-Situationen verwendet werden, um nicht mehr benötigte Rückmeldungungen aus dem Eingabepuffer zu entfernen
Delphi:	function LS_FlushBuffer(AValue: Integer): Integer; function LSX_FlushBuffer(LSID: Integer; AValue: Integer): Integer;
C++:	int FlushBuffer (int lAValue);
LabView:	
Parameter:	AValue: momentan nicht verwendet, kann =0 gesetzt werden
Beispiel:	LS.FlushBuffer(0);

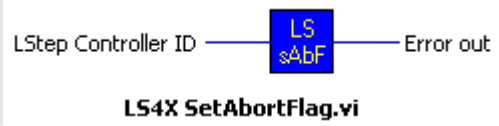
LSX_FreeLSID (nur LSTEP4X-API)	
Beschreibung	Gibt eine erzeugte LStep-ID-Nummer wieder frei. Diese wird als zusätzlicher Parameter bei den LSTEP4X-API- Befehlen verwendet, um aus mehreren angeschlossenen LSteps die LStep zu wählen, auf die sich der Befehl beziehen soll. FreeLSID sollte erst nach Disconnect aufgerufen werden.
Delphi	function LSX_FreeLSID(LSID: Integer): Integer;
C++	-
LabView:	
Parameter	LSID: freizugebende LStep-ID-Nummer; diese darf nach FreeLSID nicht mehr verwendet werden
Beispiel	var LStep1: Integer; ... LSX_CreateLSID(&LStep1); LSX_ConnectSimple(LStep1, ...); ... LSX_Disconnect(LStep1); LSX_FreeLSID(LStep1);

LS_LoadConfig	
Beschreibung:	LSTEP-Konfiguration (Schnittstelle, Achseinstellungen, Regler) aus INI-Datei laden. Das Format der INI-Datei ist mit der Win-Commander-INI-Datei kompatibel, d.h. die Einstellungen können aus dem Win-Commander (Wincom4.ini) übernommen werden. Die geladene Konfiguration wird in den Funktionen LS_Connect und LS_SetControlPars verwendet.
Delphi:	function LS_LoadConfig(FileName: PChar): Integer; function LSX_LoadConfig(LSID: Integer; FileName: PChar): Integer;
C++:	int LoadConfig (char *pcFileName);
LabView:	
Parameter:	FileName: Dateiname der INI-Datei als nullterminierter String
Beispiel:	LS.LoadConfig("C:\LStepTest\LStep.INI");

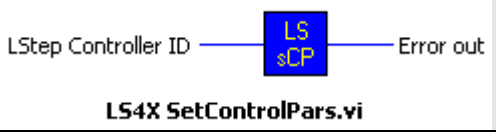
LS_SaveConfig	
Beschreibung:	LSTEP-Konfiguration (Schnittstelle, Achseinstellungen, Regler) in INI-Datei speichern. Das Format der INI-Datei ist mit der Win-Commander-INI-Datei kompatibel.
Delphi:	function LS_SaveConfig(FileName: PChar): Integer; function LSX_SaveConfig(LSID: Integer; FileName: PChar): Integer;
C++:	int SaveConfig (char *pcFileName);
LabView:	
Parameter:	FileName: Dateiname der INI-Datei als nullterminierter String
Beispiel:	LS.SaveConfig("C:\LStepTest\LStep.INI");

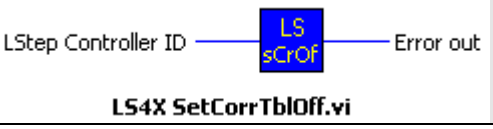
LS_SendString		
Beschreibung:	String an LSTEP senden	
Delphi:	function LS_SendString(Str, Ret: PChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer; function LSX_SendString(LSID: Integer; Str, Ret: PChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer;	
C++:	int SendString (char *pcStr,char *pcRet,int lMaxLen,BOOL ReadLine,int lTimeOut);	
LabView:		
Parameter:	Str	➔ nullterminierter String, der an die Steuerung gesendet werden soll.
	Ret	➔ Puffer, der die Rückmeldung der LSTEP enthält, falls ReadLine = true
	MaxLen	➔ Maximale Anzahl von Zeichen, die in den Puffer kopiert werden dürfen.
	ReadLine	➔ Rückmeldung der LSTEP lesen:
	TimeOut	➔ maximale Wartezeit auf Rückmeldung [ms]
Beispiel:	LS.SendString("?ver\r", pcLStepVer, 256, true, 1000); // Versionsnummer lesen, Timeout 1s	

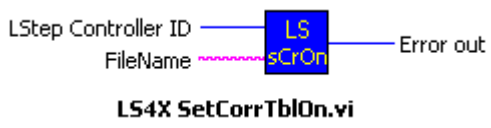
LS_SendStringPosCmd	
Beschreibung	Verfahrbefehl, welcher Rückmeldung erwartet, als String an LSTEP senden
Delphi	function LS_SendStringPosCmd(Str, Ret: PChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer; function LSX_SendStringPosCmd(LSID: Integer; Str, Ret: PChar; MaxLen: Integer; ReadLine: LongBool; TimeOut: Integer): Integer;
C++	int SendStringPosCmd (char *pcStr, char *pcRet, int lMaxLen, BOOL bReadLine, int lTimeOut);
LabView:	
Parameter:	Str ➔ nullterminierter String, der an die Steuerung gesendet werden soll.
	Ret ➔ Puffer, der die Rückmeldung der LSTEP enthält, falls ReadLine = true
	MaxLen ➔ Maximale Anzahl von Zeichen, die in den Puffer kopiert werden dürfen.
	ReadLine ➔ Rückmeldung der LSTEP lesen:
	TimeOut ➔ maximale Wartezeit auf Rückmeldung [ms]
Beispiel	LS.SendStringPosCmd("!moa 1 2\r", pcLStepVer, 256, true, 100000);

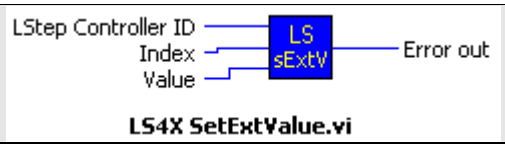
LS_SetAbortFlag	
Beschreibung:	Flag setzen, damit die Kommunikation mit der LSTEP abgebrochen wird
	Eine Funktion, die bei Aufruf von LS_SetAbortFlag noch auf eine Rückmeldung der Steuerung warten (z.B. Verfahrbefehle), kehrt dann mit einer Fehlermeldung zurück. Die Verwendung dieser Funktion ist insbesondere bei Programmen mit Botschaftsbehandlungsroutinen oder mehreren Threads sinnvoll, falls z. B. schnell eine Verfahrbewegung abgebrochen werden soll.
Delphi:	function LS_SetAbortFlag: Integer; function LSX_SetAbortFlag(LSID: Integer): Integer;
C++:	int SetAbortFlag ();
LabView:	
Parameter:	-
Beispiel:	LS.SetAbortFlag(); LS.StopAxes(); (Kommunikation mit der LSTEP abbrechen und das Kommando zum Stoppen aller Achsen senden)

LS_SetCommandTimeout	
Beschreibung:	Setzt die Timeoutzeiten für das Warten auf Rückmeldung, Positionieren und Kalibrieren.
Delphi:	function LS_ SetCommandTimeout (AtoRead, AtoMove, AtoCalibrate: Integer): Integer; LSX_SetCommandTimeout(LSID: Integer; AtoRead, AtoMove, AtoCalibrate: Integer): Integer;
C++:	int SetCommandTimeout (int lAtoRead, int lAtoMove, int lAtoCalibrate) ;
LabView:	
Parameter:	AtoRead: Timeoutzeit für das Warten auf Rückmeldung [ms] AtoMove: Timeoutzeit für Positionieren [ms] AtoCalibrate: Timeoutzeit für Kalibrieren [ms]
Beispiel:	LS. SetCommandTimeout (int lAtoRead, int lAtoMove, int lAtoCalibrate) ;

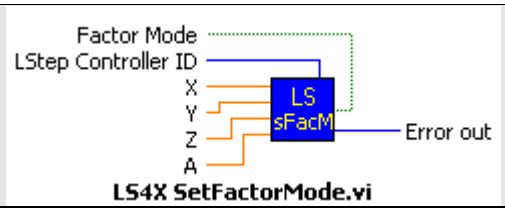
LS_SetControlPars	
Beschreibung:	Überträgt die mit LS_LoadConfig geladenen Parameter an die LSTEP.
Delphi:	function LS_SetControlPars: Integer; function LSX_SetControlPars(LSID: Integer): Integer;
C++:	int SetControlPars ();
LabView:	
Parameter:	-
Beispiel:	LS.SetControlPars();

LS_SetCorrTblOff	
Beschreibung	Achsenkorrektur deaktivieren
Delphi	function LS_SetCorrTblOff: Integer; function LSX_SetCorrTblOff(LSID: Integer): Integer;
C++	int SetCorrTblOff ();
LabView:	
Parameter	-
Beispiel	LS.SetCorrTblOff() ;

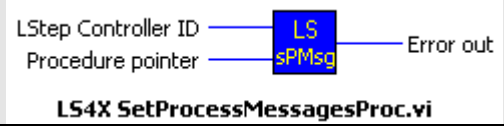
LS_SetCorrTblOn	
Beschreibung	Achsenkorrektur in x/y-Matrix mit linearer Interpolation aktivieren
Delphi	function LS_SetCorrTblOn(AFileName: PChar): Integer; function LSX_SetCorrTblOn(LSID: Integer; AFileName: PChar): Integer;
C++	int SetCorrTblOn (char *pcAFileName);
LabView:	
Parameter	Die Korrekturtabelle wird manuell in eine Ini-Datei eingegeben. Der Dateiname dieser Ini-Datei wird durch AFileName angegeben. Aufbau der Korrekturtabelle: In der Sektion [Options] wird die Achsenkorrektur mit lin. Interpolation durch die Zeile „CorrectionXY=1“ aktiviert. XCount und YCount geben die Anzahl der Korrekturwerte an. Der Parameter XDistance bestimmt den Abstand der Meßpunkte in einer Reihe (X-Achse), YDistance den Abstand der Reihen (Y-Achse). Die Sektion [CorrTbl] enthält die Korrekturwerte. Jeder Soll-Position (x/y-Wertepaar) wird eine korrigierte Position zugeordnet, wobei die Soll-Positionen immer einer der Punkte in dem durch XCount, YCount, XDistance und YDistance festgelegten Raster sein müssen. Die Zuordnungen (Soll-Position=Korrigierte Position) können in der Korrekturtabelle in beliebiger Reihenfolge erfolgen, wichtig ist nur, daß die Soll-Positionen immer in diesem Raster liegen (Der Nullpunkt der Korrekturtabelle ist (0|0)). Beispiel einer Korrekturtabelle: [Options] CorrectionXY=1 XCount=3 YCount=3 XDistance=1.0 YDistance=1.0 [CorrTbl] 0.0 0.0=0.0 0.0 1.0 0.0=1.0 0.0 2.0 0.0=2.0 0.0 0.0 1.0=0.0 1.0 1.0 1.0=0.9 1.1 (Soll-Position x=1 y=1, korrigierte Position x=0.9 y=1.1) 2.0 1.0=2.0 1.0 0.0 2.0=0.0 2.0 1.0 2.0=1.0 2.0 2.0 2.0=2.0 2.0
Beispiel	LS.SetCorrTblOn(„C:\...\corrtbl.ini“);

LS_SetExtValue	
Beschreibung:	Schaltet Erweiterungen des API ein, teilweise handelt es sich dabei um experimentelle Modi zu Debugging-Zwecken
Delphi:	function LS_SetExtValue(AName: Integer; AValue: Integer): Integer; function LSX_SetExtValue(LSID: Integer; AName, AValue: Integer): Integer;
C++:	int SetExtValue (int lAName, int lAValue);
LabView:	
Parameter:	AName: Nummer der erweiterten Funktion AValue: Parameter **AName=2 (IFSleepTime)** stellt das Polling-Intervall für das DPRAM der LStep-PCI ein AValue: Zeit-Intervall in [ms], Standard ist 10 **AName=3 (ProtMoveOnly)** schaltet Filter für Log-Datei ein, durch welches nur Moves&Fehler protokolliert werden AValue=1 → Filter an AValue=0 → Filter aus **AName=4 (Max_LogLn)** begrenzt die Länge der Log-Datei, ältere Log-Datei wird in .old umbenannt AValue=Maximale Zeilenzahl **AName=5 (ThreadPriority)** ändert die Priority der Threads des LStep API. Nach Connect werden die Threads immer auf normale Priorität gesetzt, mit SetExtValue(5, ...) kann dies im nachhinein geändert werden. AValue=Windows-API-Konstante für Thread-Priorität wie THREAD_PRIORITY_ABOVE_NORMAL

Beispiel:	LS.SetExtValue(3, 1); // Filter für Move-Befehle an
	LS.SetExtValue(4, 10000); // maximale Länge der Log-Datei = 10000 Zeilen
	LS.SetExtValue(5, THREAD_PRIORITY_HIGHEST);

LS_SetFactorMode	
Beschreibung	Positionswert-Umrechnung für ‚krumme' Spindelsteigungen
Delphi	function LS_SetFactorMode(AFactorMode: LongBool; X, Y, Z, A: Double): Integer; function LSX_SetFactorMode(LSID: Integer; AFactorMode: LongBool; X, Y, Z, A: Double): Integer;
C++	int SetFactorMode (BOOL bAFactorMode, double dX, double dY, double dZ, double dA);
LabView:	
Parameter	AFactorMode: Faktor-Modus einschalten Der Befehl aktiviert eine API-interne Umrechnung der Positionswerte/Spindelsteigung, um so bei 'krummen' Spindelsteigungen Rundungsfehler zu vermeiden X, Y, Z, R: Spindelsteigungswerte, die an die LStep übertragen werden (möglichst Werte wie 1.0 oder 4.0, sodass ein Mikrostep einem nichtperiodischen Dezimalbruch entspricht) Erst **nach** SetFactorMode sollte dann SetPitch mit der tatsächlichen, physikalischen Spindelsteigung aufgerufen werden. Alle Verfahrbefehle verwenden nach Aufruf von SetFactorMode und SetPitch eine Faktor-Umrechnung, damit die LStep korrekt positioniert. an LStep gesendeter Positionsvektor = Positionsvektor * an LStep gesendete Spindelsteigung / physikalische Spindelsteigung
Beispiel	LS.SetFactorMode(true, 1, 1, 1, 0); LS.SetPitch(1.234, 1.234, 2.345, 0); LS.MoveAbs(1.234, 2.468, 2.345, 0, true);

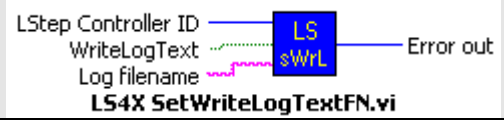
LS_SetLanguage	
Beschreibung:	Sprachumschaltung LSTEP-API (Protokoll/Meldungen)
Delphi:	function LS_SetLanguage(PLN: PChar): Integer; function LSX_SetLanguage(LSID: Integer; PLN: PChar): Integer;
C++:	int SetLanguage (char *pcPLN);
LabView:	
Parameter:	PLN: Sprache (Kürzel, z.B. „DEU„ oder „ENG„) Die entsprechende Textdatei (LSTEP4deu.txt oder LSTEP4eng.txt) muß im Verzeichnis des Programms liegen
Beispiel:	LS.SetLanguage('ENG');

LS_SetProcessMessagesProc	
Beschreibung	Ermöglicht das Ersetzen der internen Message-Dispatching Prozedur des LStep API. Das LStep API verarbeitet während des Wartens auf Rückmeldungen der LStep im Main-Thread Messages. Wenn sie das Message-Dispatching abschalten wollen oder durch eigenen Code ersetzen wollen, können sie SetProcessMessagesProc zum Verwenden einer Callback-Prozedur verwenden.
Delphi	function LS_SetProcessMessagesProc(Proc: Pointer): Integer; function LSX_SetProcessMessagesProc(LSID: Integer; Proc: Pointer): Integer;
C++	int SetProcessMessagesProc (void* pProc);
LabView:	
Parameter	pProc muss ein Zeiger auf eine stdcall-Prozedur ohne Parameter sein: void MyProcessMessages () { .. }
Beispiel	LS. SetProcessMessagesProc (&MyProcessMessages);

LS_SetShowCmdList	
Beschreibung:	LStep-API Befehlsliste Ein/Aus
Delphi:	function LS_SetShowCmdList(ShowCmdList: LongBool): Integer; function LSX_SetShowCmdList(LSID: Integer; ShowCmdList: LongBool): Integer;
C++:	int SetShowCmdList (BOOL bShowCmdList);
LabView:	-
Parameter:	ShowProt: Gibt an, ob das Fenster „LStep-API Befehlsliste“ gezeigt werden soll
Beispiel:	LS.SetShowCmdList(true); // Schnittstellen-Protokoll zeigen falls nicht bereits sichtbar

LS_SetShowProt	
Beschreibung:	Schnittstellen-Protokoll Ein/Aus
Delphi:	function LS_SetShowProt(ShowProt: LongBool): Integer; function LSX_SetShowProt(LSID: Integer; ShowProt: LongBool): Integer;
C++:	int SetShowProt (BOOL ShowProt);
LabView:	
Parameter:	ShowProt: Gibt an, ob das Fenster „Schnittstellen-Protokoll“ gezeigt werden soll
Beispiel:	LS.SetShowProt(true); // Schnittstellen-Protokoll zeigen falls nicht bereits sichtbar

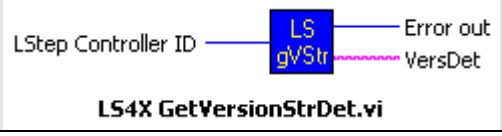
LS_SetWriteLogText	
Beschreibung:	Schreiben der Protokoll-Datei LSTEP4.log ein-/ausschalten (Standardmäßig ist das Schreiben in LSTEP4.log ausgeschaltet)
Delphi:	function LS_SetWriteLogText(AWriteLogText: LongBool): Integer; function LSX_SetWriteLogText(LSID: Integer; AWriteLogText: LongBool): Integer;
C++:	int SetWriteLogText (BOOL AWriteLogText);
LabView:	
Parameter:	-
Beispiel:	LS.SetWriteLogText (true);

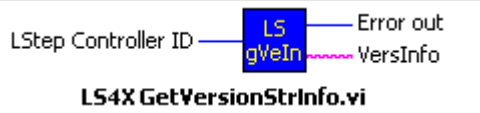
LS_SetWriteLogTextFN	
Beschreibung	Schreiben des Schnittstellen-Protokolls in eine bestimmte Datei ein-/ausschalten (Standardmäßig ist das Schreiben ausgeschaltet)
Delphi	function LS_SetWriteLogTextFN(AWriteLogText: LongBool; ALogFN: PChar): Integer; function LSX_SetWriteLogTextFN(LSID: Integer; AWriteLogText: LongBool; ALogFN: PChar): Integer;
C++	int SetWriteLogTextFN (BOOL bAWriteLogText, char *pcALogFN);
LabView:	
Parameter	AWriteLogText: true => Protokolldatei schreiben ALogFN: Dateiname der Protokolldatei
Beispiel	LS.SetWriteLogTextFN(true, „C:\Temp\prot.txt“);

Steuerungs-Info:

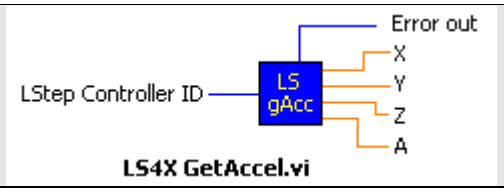
LS_GetSerialNr	
Beschreibung:	Seriennummer der Steuerung auslesen
Delphi:	function LS_GetSerialNr(SerialNr: PChar; MaxLen: Integer): Integer; function LSX_GetSerialNr(LSID: Integer; SerialNr: PChar; MaxLen: Integer): Integer;
C++:	int GetSerialNr (char *pcSerialNr,int lMaxLen);
LabView:	
Parameter:	SerialNr: Zeiger auf einen Puffer, in dem die Seriennummer zurückgegeben wird MaxLen: Maximale Anzahl von Zeichen, die in den Puffer kopiert werden dürfen
Beispiel:	LS.GetSerialNr(pcSerialNr, 256);

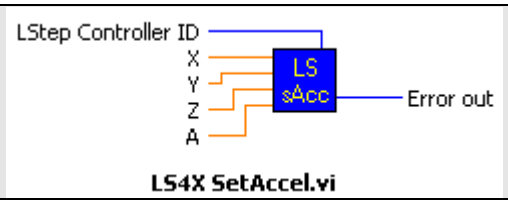
LS_GetVersionStr	
Beschreibung:	liefert die aktuelle Versionsnummer der Firmware zurück
Delphi:	function LS_GetVersionStr(Vers: PChar; MaxLen: Integer): Integer; function LSX_GetVersionStr(LSID: Integer; Vers: PChar; MaxLen: Integer): Integer;
C++:	int GetVersionStr (char *pcVers,int lMaxLen);
LabView:	
Parameter:	Stat: Zeiger auf einen Puffer, in dem der Versions-String zurückgegeben wird MaxLen: Maximale Anzahl von Zeichen, die in den Puffer kopiert werden dürfen
Beispiel:	LS.GetVersionStr(pcVers, 64); // Versionsnummer auslesen

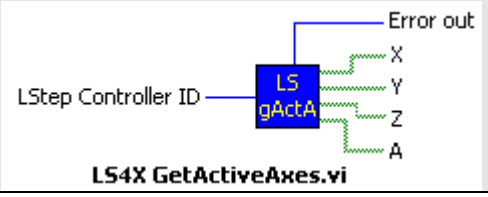
LS_GetVersionStrDet	
Beschreibung:	Detaillierte Versionsnummer der Firmware auslesen
Delphi:	function LS_GetVersionStrDet(VersDet: PChar; MaxLen: Integer): Integer; function LSX_GetVersionStrDet(LSID: Integer; VersDet: PChar; MaxLen: Integer): Integer;
C++:	int GetVersionStrDet (char *pcVersDet, int lMaxLen);
LabView:	
Parameter:	VersDet: Zeiger auf einen Puffer, in dem der detaillierteVersions-String zurückgegeben wird MaxLen: Maximale Anzahl von Zeichen, die in den Puffer kopiert werden dürfen
Beispiel:	LS.GetVersionStrDet(pcVersDet, 64); // detaillierte Versionsnummer auslesen

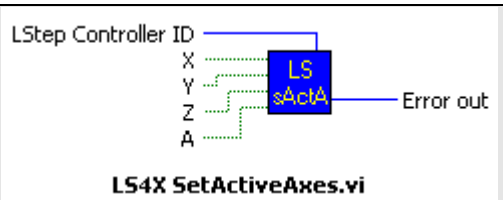
LS_GetVersionStrInfo	
Beschreibung:	Liefert detaillierte Informationen zur Versionsnummer
Delphi:	function LS_ GetVersionStrInfo (VersInfo: PChar; MaxLen: Integer): Integer; function LSX_ GetVersionStrInfo (LSID: Integer; VersInfo: PChar; MaxLen: Integer): Integer;
C++:	int GetVersionStrInfo (char *pcVersInfo, int lMaxLen);
LabView:	
Parameter:	VersInfo: Zeiger auf einen Puffer, in dem Wochentag.Kalenderwoche.Jahr-fortlaufende Nummer zurückgegeben werden. z. B.: T04.35.02-0004 MaxLen: Maximale Anzahl von Zeichen, die in den Puffer kopiert werden dürfen
Beispiel:	LS.GetVersionStrInfo (pcVersInfo, 64);

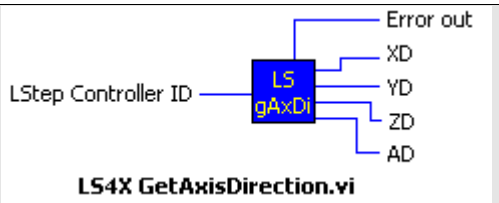
Einstellungen

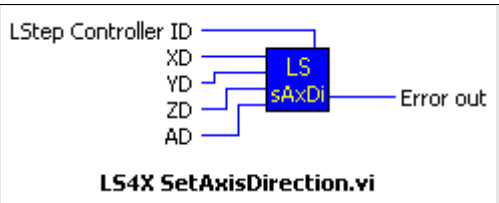
LS_GetAccel	
Beschreibung:	Beschleunigung abfragen
Delphi:	function LS_GetAccel(var X, Y, Z, R: Double): Integer; function LSX_GetAccel(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetAccel (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Beschleunigungswerte [m/s²]
Beispiel:	LS.GetAccel(&X, &Y, &Z, &A);

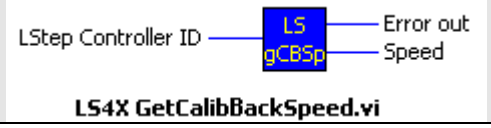
LS_SetAccel	
Beschreibung:	Beschleunigung einstellen
Delphi:	function LS_SetAccel(X, Y, Z, R: Double): Integer; function LSX_SetAccel(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetAccel(double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z und A 0.01 – 10.00 [m/s²]
Beispiel:	LS.SetAccel(1.0, 1.5, 0, 0);

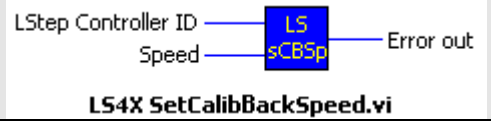
LS_GetActiveAxes	
Beschreibung:	Liefert die Achsenfreigabe
Delphi:	function LS_GetActiveAxes(var Flags: Integer): Integer; function LSX_GetActiveAxes(LSID: Integer; var Flags: Integer): Integer;
C++:	int GetActiveAxes (int *plFlags);
LabView:	
Parameter:	Flags: 32-bit-Integer, welcher nach Aufruf der Funktion in den Bits 0-4 die Bit-Maske enthält. Bit 0 = 1 → X-Achse freigeschaltet Bit 2 = 0 → Z-Achse nicht freigeschaltet
Beispiel:	LS.GetActiveAxes(&Flags);

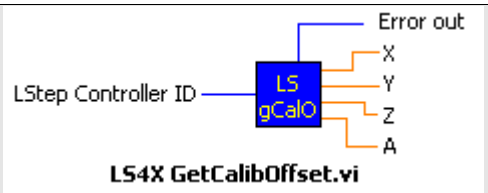
LS_SetActiveAxes	
Beschreibung:	Achsenfreigabe
Delphi:	function LS_SetActiveAxes(Flags: Integer): Integer; function LSX_SetActiveAxes(LSID: Integer; Flags: Integer): Integer;
C++:	int SetActiveAxes(int Flags);
LabView:	
Parameter:	Flags: Bit-Maske Bit 0 = 1 → X-Achse freigeschaltet Bit 2 = 0 → Z-Achse nicht freigeschaltet
Beispiel:	LS.SetActiveAxes(3); /* X- und Y-Achse freigeben (Bits 0 u. 1 gesetzt), Z-Achse nicht freigegeben (Bit 2 = 0) */

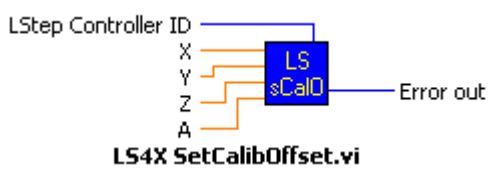
LS_GetAxisDirection	
Beschreibung	Drehrichtungs-Umkehr abfragen
Delphi	function LS_GetAxisDirection(var XD, YD, ZD, AD: Integer): Integer; function LSX_GetAxisDirection(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;
C++	int GetAxisDirection (int *plXD, int *plYD, int *plZD, int *plAD);
LabView:	
Parameter	XD, YD, ZD, AD: 32-bit-Integers 0 => normale Drehrichtung 1 => Drehrichtungs-Umkehr
Beispiel	LS.GetAxisDirection(&XD, &YD, &ZD, &AD);

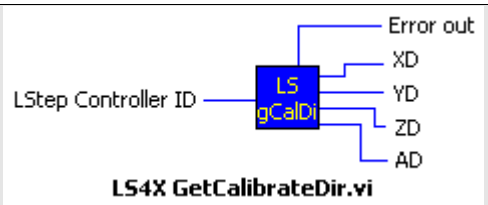
LS_SetAxisDirection	
Beschreibung	Drehrichtungs-Umkehr
Delphi	function LS_SetAxisDirection(XD, YD, ZD, AD: Integer): Integer; function LSX_SetAxisDirection(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;
C++	int SetAxisDirection (int lXD, int lYD, int lZD, int lAD);
LabView:	
Parameter	XY, YD, ZD, AD: 0 => normale Drehrichtung 1 => Drehrichtungs-Umkehr
Beispiel	LS.SetAxisDirection(1, 0, 0, 0); // Drehrichtung der X-Achse umkehren

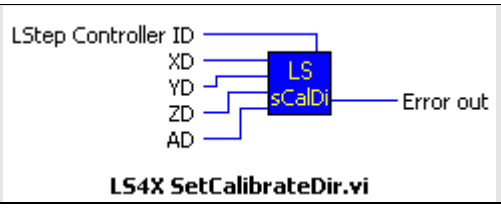
LS_GetCalibBackSpeed	
Beschreibung:	Liest die Umdrehungsgeschwindigkeit, mit der die Achsen beim Kalibrieren nach dem Anfahren der Endschalter wieder herausgefahren werden. Die Geschwindigkeit entspricht dem ausgegebenen Wert * 0.01 U/s.
Delphi:	function LS_GetCalibBackSpeed(var ISpeed: Integer): Integer; function LSX_ GetCalibBackSpeed (LSID: Integer; var ISpeed: Integer): Integer;
C++:	int GetCalibBackSpeed (int *plSpeed);
LabView:	
Parameter:	lSpeed: Geschwindigkeitswert
Beispiel:	LS. GetCalibBackSpeed (&lSpeed);

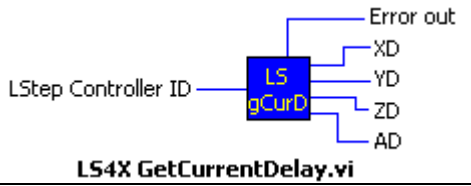
LS_SetCalibBackSpeed	
Beschreibung:	Setzt die Umdrehungsgeschwindigkeit, mit der die Achsen beim Kalibrieren nach dem Anfahren der Endschalter wieder herausgefahren werden.Die Geschwindigkeit entspricht dem angegebenen Wert*0.01 U/s.
Delphi:	function LS_SetCalibBackSpeed(ISpeed: Integer): Integer; function LSX_ SetCalibBackSpeed (LSID: Integer; ISpeed: Integer): Integer;
C++:	int SetCalibBackSpeed (int lSpeed);
LabView:	
Parameter:	lSpeed: Geschwindigkeit, Wertebereich 5 bis 100
Beispiel:	LS. SetCalibBackSpeed (10); //Die Endschalter werden bei der Kalibrierung nach dem Anfahren mit 0.1 U/s verlassen.

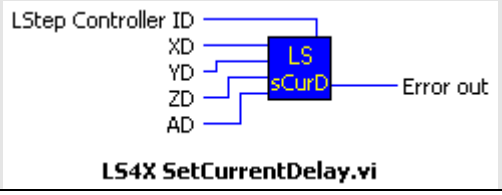
LS_GetCalibOffset	
Beschreibung:	Kalibrier-Offset abfragen
Delphi:	function LS_GetCalibOffset(var X, Y, Z, A: Double): Integer; function LSX_GetCalibOffset(LSID: Integer; var X, Y, Z, R: Double): Integer;
C++:	int GetCalibOffset (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z, A: Kalibrier-Offset, abhängig von Dimension.
Beispiel:	LS.GetCalibOffset(&X, &Y, &Z, &A);

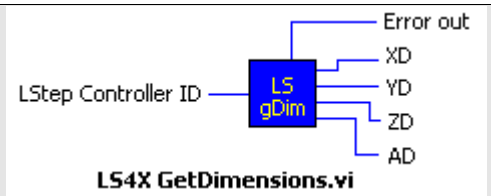
LS_SetCalibOffset	
Beschreibung:	Kalibrier-Offset
Delphi:	function LS_SetCalibOffset(X, Y, Z, A: Double): Integer; function LSX_SetCalibOffset(LSID: Integer; X, Y, Z, R: Double): Integer;
C++:	int SetCalibOffset (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, y, z und a 0 – 32*50000 (32*Spindelsteigung)
Beispiel:	LS.SetCalibOffset(1, 1, 1, 1); (Die Achsen X, Y und Z werden beim Kalibrieren jeweils 1mm (bei Dim 2 2 2) vom Nullendschalter in Richtung Tischmitte verfahren und dann die Position Null gesetzt (Softwaregrenze).)

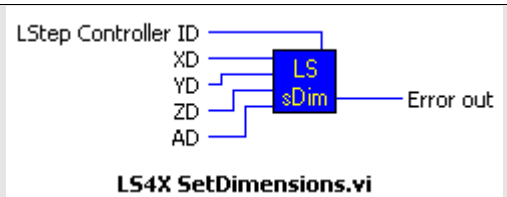
LS_GetCalibrateDir	
Beschreibung	Vorzeichen-Umkehr bei Kalibration abfragen
Delphi	function LS_GetCalibrateDir(var XD, YD, ZD, AD: Integer): Integer; function LSX_GetCalibrateDir(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;
C++	int GetCalibrateDir (int *plXD, int *plYD, int *plZD, int *plAD);
LabView:	
Parameter	XD, YD, ZD, AD: 32-bit-Integers 0 => keine Vorzeichen-Umkehr 1 => Vorzeichen-Umkehr
Beispiel	LS.GetCalibrateDir(&XD, &YD, &ZD, &AD);

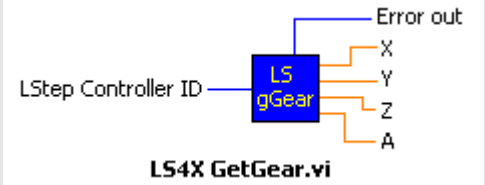
LS_SetCalibrateDir	
Beschreibung	Vorzeichen-Umkehr bei Kalibration
Delphi	function LS_SetCalibrateDir(XD, YD, ZD, AD: Integer): Integer; function LSX_SetCalibrateDir(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;
C++	int SetCalibrateDir (int lXD, int lYD, int lZD, int lAD);
LabView:	
Parameter	XD, YD, ZD, AD: 0 => keine Vorzeichen-Umkehr 1 => Vorzeichen-Umkehr
Beispiel	LS.SetCalibrateDir(1, 1, 0, 0);

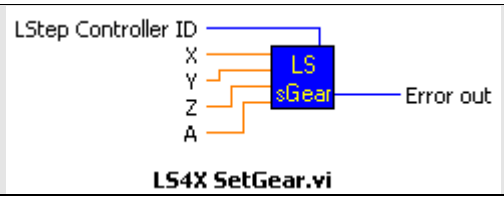
LS_GetCurrentDelay	
Beschreibung:	Gibt an Zeitverzögerung für die Stromabsenkung
Delphi:	function LS_GetCurrentDelay(var X, Y, Z, R: Integer): Integer; function LSX_GetCurrentDelay(LSID: Integer; var X, Y, Z, R: Integer): Integer;
C++:	int GetCurrentDelay (int *plX, int *plY, int *plZ, int *plR);
LabView:	
Parameter :	X, Y, Z, R: Zeitverzögerung in ms
Beispiel:	LS.SetCurrentDelay(&X, &Y, &Z, &A);

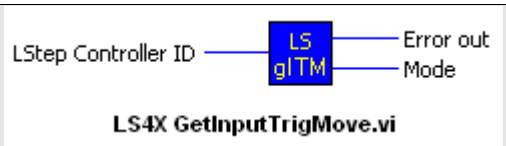
LS_SetCurrentDelay	
Beschreibung:	Zeitverzögerung für die Stromabsenkung
Delphi:	function LS_SetCurrentDelay(X, Y, Z, R: Integer): Integer; function LSX_SetCurrentDelay(LSID: Integer; X, Y, Z, R: Integer): Integer;
C++:	int SetCurrentDelay (int lX, int lY, int lZ, int lR);
LabView:	
Parameter:	X, Y, Z, R: 0-10000 [ms]
Beispiel:	LS.SetCurrentDelay(100, 300, 1000, 0) ;

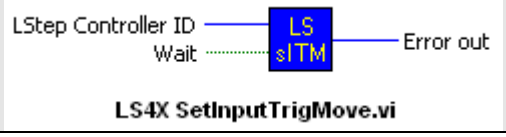
LS_GetDimensions	
Beschreibung:	Abfrage der Dimensionen der Achsen
Delphi:	function LS_GetDimensions(var XD, YD, ZD, AD: Integer): Integer; function LSX_GetDimensions(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;
C++:	int GetDimensions (int *plXD, int *plYD, int *plZD, int *plAD);
LabView:	
Parameter:	XD, YD, ZD, AD: Dimensionswerte
	0 ➔ Microsteps 1 ➔ µm 2 ➔ Millimeter 3 ➔ Grad 4 ➔ Umdrehungen
Beispiel:	LS. GetDimensions (&XD, &YD, &ZD, &AD);

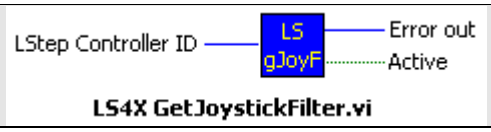
LS_SetDimensions	
Beschreibung:	Dimensionen der Achsen einstellen
Delphi:	function LS_SetDimensions(XD, YD, ZD, AD: Integer): Integer; function LSX_SetDimensions(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;
C++:	int SetDimensions (int lXD,int lYD,int lZD,int lAD);
LabView:	
Parameter:	Dimension von X, Y, Z und A-Achse:
	0 ➔ Microsteps 1 ➔ µm 2 ➔ Millimeter 3 ➔ Grad 4 ➔ Umdrehungen
Beispiel:	LS.SetDimensions(3, 2, 2); // X-Achse in Grad; Y und Z in mm

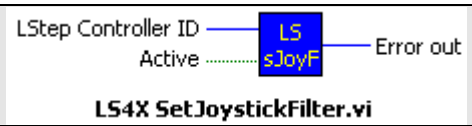
LS_GetGear	
Beschreibung:	Getriebe-Übersetzung abfragen
Delphi:	function LS_GetGear(var X, Y, Z, A: Double): Integer; function LSX_GetGear(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetGear (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Getriebe-Übersetzungswerte
Beispiel:	LS. GetGear (&X, &Y, &Z, &A);

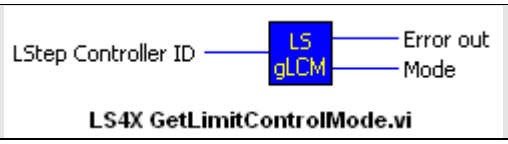
LS_SetGear	
Beschreibung:	Getriebe-Übersetzung programmieren
Delphi:	function LS_SetGear(X, Y, Z, A: Double): Integer; function LSX_SetGear(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetGear (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z und A 0.01 – 1000
Beispiel:	LS.SetGear(4.0, 2.0, 1.0, 1.0); /* Getriebe-Übersetzungen ¼ bei Z, ½ bei Y und 1/1 bei Z u. A werden programmiert */

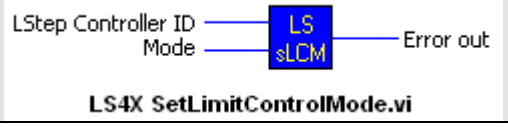
LS_GetInputTrigMove	
Beschreibung:	Liefert die Konfiguration vom Pin1 auf dem MFP.
Delphi:	function LS_GetInputTrigMove (var Mode: Integer): Integer; function LSX_GetInputTrigMove (LSID: Integer; var Mode: Integer): Integer;
C++:	int GetInputTrigMove (int *plMode);
LabView:	
Parameter:	**lMode** – Modus. lMode = 0 → Funktion nicht aktiv lMode = 1 → absolut positionieren bei positiver Flanke lMode = 2 → absolut positionieren bei negativer Flanke lMode = 3 → relativ positionieren bei positiver Flanke lMode = 4 → relativ positionieren bei negativer Flanke
Beispiel:	LS. GetInputTrigMove (&lMode);

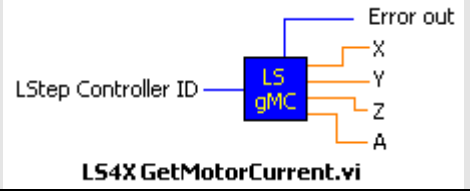
LS_SetInputTrigMove	
Beschreibung:	Konfiguriert den Pin 1 auf dem MFP, so dass man mit einem externen Signal einen Move starten kann.
Delphi:	function LS_SetInputTrigMove (Mode: Integer; Wait: LongBool): Integer; function LSX_SetInputTrigMove (LSID: Integer; Mode: Integer; Wait: LongBool): Integer;
C++:	int SetInputTrigMove (int lMode, BOOL bWait);
LabView:	
Parameter:	**lMode** – Modus. lMode = 0 → Funktion nicht aktiv lMode = 1 → absolut positionieren bei positiver Flanke lMode = 2 → absolut positionieren bei negativer Flanke lMode = 3 → relativ positionieren bei positiver Flanke lMode = 4 → relativ positionieren bei negativer Flanke Verfahren wird der Wert, der in „distance“ steht. **bWait** – das Warten auf einen Move. bWait = 1 → es wird so lange gewartet, bis nach einem externen Signal ein Move ausgeführt wird, danach wird der Modus auf 0 gesetzt. bWait = 0 → es wird nicht auf einen Move gewartet. bWait wird nicht ausgewertet, wenn lMode = 0.
Beispiel:	LS. SetInputTrigMove (3, False);

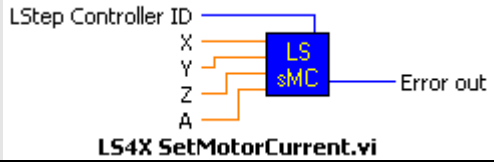
LS_GetJoystickFilter	
Beschreibung:	Gibt an, ob die Filterung und Hysterese im Joystick-Betrieb aktiviert ist.
Delphi:	function LS_GetJoystickFilter(var bActive: LongBool): Integer; function LSX_ GetJoystickFilter (LSID: Integer; var bActive: LongBool): Integer;
C++:	int GetJoystickFilter (BOOL *pbActive);
LabView:	
Parameter:	bActive: True – Filterung aktiviert False – deaktiviert
Beispiel:	LS. SetJoystickFilter (&Active);

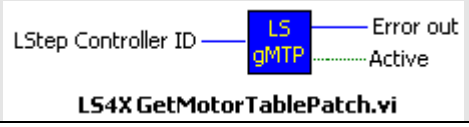
LS_SetJoystickFilter	
Beschreibung:	Aktivierung/Deaktivierung der Filterung und Hysterese im Joystick-Betrieb.
Delphi:	function LS_SetJoystickFilter(bActive: LongBool): Integer; function LSX_ SetJoystickFilter (LSID: Integer; bActive: LongBool): Integer;
C++:	int SetJoystickFilter (BOOL bActive);
LabView:	
Parameter:	bActive: True – Aktivierung der Filterung False – Deaktivierung
Beispiel:	LS. SetJoystickFilter (True);

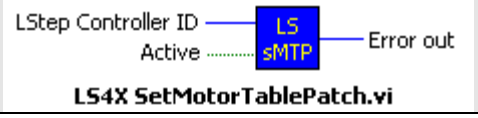
LS_GetLimitControlMode	
Beschreibung:	Liefert den Modus für die Überwachung der Softwarelimits.
Delphi:	function LS_GetLimitControlMode (var Mode: Integer): Integer; function LSX_GetLimitControlMode (LSID: Integer; var Mode: Integer): Integer;
C++:	int GetLimitControlMode (int *plMode);
LabView:	
Parameter:	**lMode** – Modus. lMode = 0 → Moves, die außerhalb des Verfahrbereiches liegen, werden nur bis zu den Grenzen des Verfahrbereiches ausgeführt. lMode = 1 → Moves, die außerhalb des Verfahrbereiches liegen, werden nicht ausgeführt.
Beispiel:	LS. GetLimitControlMode (&lMode);

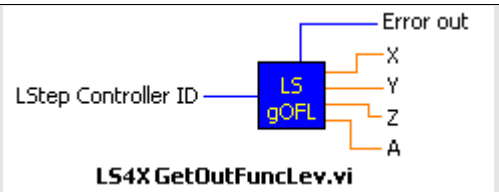
LS_SetLimitControlMode	
Beschreibung:	Setzt den Modus für die Überwachung der Softwarelimits.
Delphi:	function LS_SetLimitControlMode (Mode: Integer): Integer; function LSX_SetLimitControlMode (LSID: Integer; Mode: Integer): Integer;
C++:	int SetLimitControlMode (int lMode);
LabView:	
Parameter:	**lMode** – Modus. lMode = 0 → Moves, die außerhalb des Verfahrbereiches liegen, werden nur bis zu den Grenze des Verfahrbereiches ausgeführt. lMode = 1 → Moves, die außerhalb des Verfahrbereiches liegen, werden nicht ausgeführt.
Beispiel:	LS. SetLimitControlMode(1);

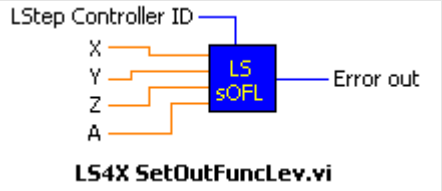
LS_GetMotorCurrent	
Beschreibung:	Motorstrom abfragen
Delphi:	function LS_GetMotorCurrent(var X, Y, Z, A: Double): Integer; function LSX_GetMotorCurrent(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetMotorCurrent (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Motorstrom [A]
Beispiel:	LS.GetMotorCurrent(&X, &Y, &Z, &A);

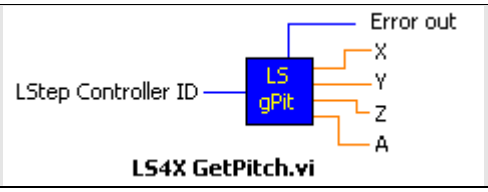
LS_SetMotorCurrent	
Beschreibung:	Motorstrom einstellen
Delphi:	function LS_SetMotorCurrent(X, Y, Z, A: Double): Integer; function LSX_SetMotorCurrent(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetMotorCurrent (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	Motorstrom X, Y, Z, A-Achse [A]
Beispiel:	LS.SetMotorCurrent(1.5, 1.5, 1.0, 1.0); // Motorstrom X u. Y 1.5 Ampere; Z u. A 1.0 Ampere

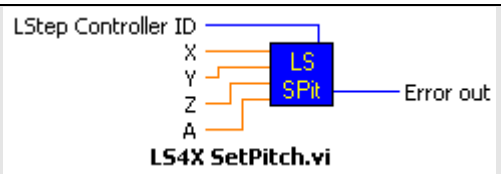
LS_GetMotorTablePatch	
Beschreibung:	Gibt an, ob die Korrekturtabelle aktiviert ist.
Delphi:	function LS_GetMotorTablePatch(bActive: var LongBool): Integer; function LSX_ GetMotorTablePatch (LSID: Integer; var bActive: LongBool): Integer;
C++:	int GetMotorTablePatch (BOOL *pbActive);
LabView:	
Parameter:	bActive: True – Tabelle ist akktiviert False – Deaktiviert
Beispiel:	LS. GetMotorTablePatch (&Active);

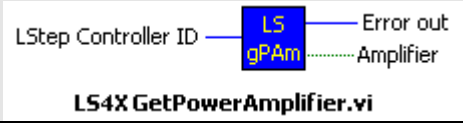
LS_SetMotorTablePatch	
Beschreibung:	Die Korrekturtabelle wird aktiviert. Die Korrekturtabelle wurde für einen Sondermotor durch Meßung ermittelt. Korrekturtabellen können auf Kundenwunsch ermittelt werden.
Delphi:	function LS_SetMotorTablePatch(bActive: LongBool): Integer; function LSX_ SetMotorTablePatch (LSID: Integer; bActive: LongBool): Integer;
C++:	int SetMotorTablePatch (BOOL bActive);
LabView:	
Parameter:	bActive: True – Aktivierung der Korrekturtabelle False – Deaktivierung
Beispiel:	LS. SetMotorTablePatch (True);

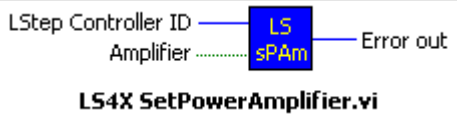
LS_GetOutFuncLev	
Beschreibung	Gibt die Geschwindigkeit an, bei der eine Umschaltung des Stroms, von parametrisiertem Strom auf maximalen Strom, erfolgt.
Delphi	function LS_GetOutFuncLev(var X, Y, Z, R: Double): Integer; function LSX_ GetOutFuncLev (LSID: Integer; var X, Y, Z, R: Double): Integer;
C++	int GetOutFuncLev (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter	X, Y, Z, R: Geschwindigkein in U/s
Beispiel	LS.GetCurrentDelay(&X, &Y, &Z, &A);

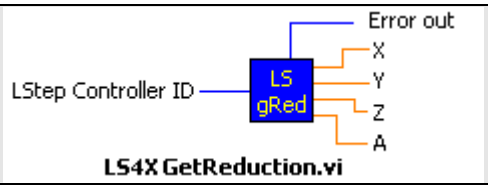
LS_SetOutFuncLev	
Beschreibung	Beim überschreiten der eingestellten Geschwindigkeit erfolgt eine Umschaltung des Stroms, von parametrisiertem Strom auf maximalen Strom.
Delphi	function LS_SetOutFuncLev(X, Y, Z, R: Double): Integer; function LSX_ SetOutFuncLev (LSID: Integer; X, Y, Z, R: Double): Integer;
C++	int SetOutFuncLev (double dX, double dY, double dZ, double dR);
LabView:	
Parameter	X, Y, Z, R: Geschwindigkein in U/s
Beispiel	LS.SetCurrentDelay(25, 25, 25, 25); /* Bei allen Achsen erfolgt eine Stromschaltung bei 25 U/s */

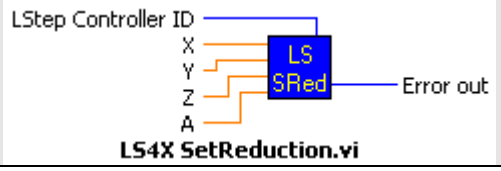
LS_GetPitch	
Beschreibung:	liefert Spindelsteigung
Delphi:	function LS_GetPitch(var X, Y, Z, R: Double): Integer; function LSX_GetPitch(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetPitch (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Spindelsteigungen [mm]
Beispiel:	LS. GetPitch (&X, &Y, &Z, &A);

LS_SetPitch	
Beschreibung:	Spindelsteigung setzen
Delphi:	function LS_SetPitch(X, Y, Z, R: Double): Integer; function LSX_SetPitch(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetPitch(double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z und A 0.001 – 68 [mm]
Beispiel:	LS.SetPitch(4, 4, 4, 4); // Spindelsteigungen aller Achsen auf 4 mm setzen

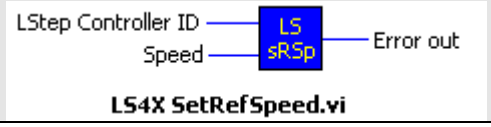
LS_GetPowerAmplifier	
Beschreibung:	Gibt an, ob die Endstufen bei LS44 ein- oder ausgeschaltet sind. Diesen Befehl gibt es nur bei LS44-Steuerung.
Delphi:	function LS_GetPowerAmplifier (bAmplifier: var LongBool): Integer; function LSX_ GetPowerAmplifier (LSID: Integer; var bAmplifier: LongBool): Integer;
C++:	int GetPowerAmplifier (BOOL *pbAmplifier);
LabView:	
Parameter:	Amplifier: True – die Endstufen sind eingeschaltet False – ausschaltet
Beispiel:	LS. GetPowerAmplifier (&Amplifier);

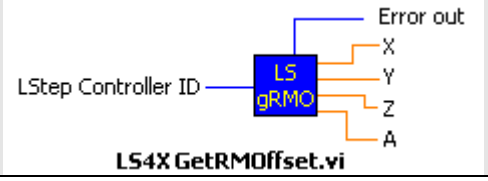
LS_SetPowerAmplifier	
Beschreibung:	Schaltet bei LS44 die Endstufen Ein/Aus. Diesen Befehl gibt es nur bei LS44-Steuerung.
Delphi:	function LS_SetPowerAmplifier (bAmplifier: LongBool): Integer; function LSX_ SetPowerAmplifier (LSID: Integer; bAmplifier: LongBool): Integer;
C++:	int SetPowerAmplifier (BOOL bAmplifier);
LabView:	
Parameter:	bAmplifier: True – Ein False – Aus
Beispiel:	LS. SetPowerAmplifier (True); // Die Endstufen einschalten

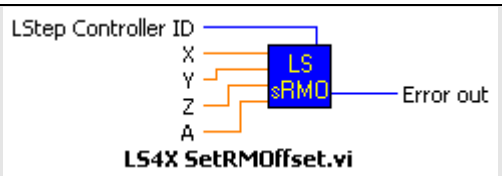
LS_GetReduction	
Beschreibung:	Stromabsenkung abfragen
Delphi:	function LS_GetReduction(var X, Y, Z, R: Double): Integer; function LSX_GetReduction(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetReduction (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Stromabsenkung
Beispiel:	LS.GetReduction(&X, &Y, &Z, &A);

LS_SetReduction	
Beschreibung:	Stromabsenkung einstellen Im Ruhezustand wird der Motornennstrom auf das parametrierte Verhältnis reduziert.
Delphi:	function LS_SetReduction(X, Y, Z, R: Double): Integer; function LSX_SetReduction(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetReduction(double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z und A 0 – 1.0
Beispiel:	LS.SetReduction(0.1, 0.7, 0.5, 0.5); /* Ruhestrom X-Achse = 0.1*Nennstrom; Y-Achse = 0.7*Nennstrom ... */

LS_GetRefSpeed	
Beschreibung:	Liest die Umdrehungsgeschwindigkeit, mit der die Achsen beim Suchen nach der Referenzmarke gefahren werden. Die Geschwindigkeit entspricht dem ausgegebenen Wert * 0.01 U/s.
Delphi:	function LS_GetRefSpeed(var lSpeed: Integer): Integer; function LSX_ GetRefSpeed (LSID: Integer; var lSpeed: Integer): Integer;
C++:	int GetRefSpeed (int *plSpeed);
LabView:	
Parameter:	lSpeed: Geschwindigkeitswert
Beispiel:	LS. GetRefSpeed (&lSpeed);

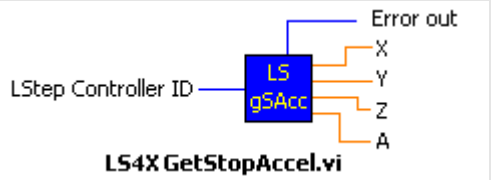
LS_SetRefSpeed	
Beschreibung:	Setzt die Umdrehungsgeschwindigkeit, mit der die Achsen beim Suchen nach der Referenzmarke gefahren werden. Die Geschwindigkeit entspricht dem angegebenen Wert * 0.01 U/s.
Delphi:	function LS_SetRefSpeed(lSpeed: Integer): Integer; function LSX_ SetRefSpeed (LSID: Integer; lSpeed: Integer): Integer;
C++:	int SetRefSpeed (int lSpeed);
LabView:	
Parameter:	lSpeed: Geschwindigkeit, Wertebereich 0 bis 100
Beispiel:	LS. SetRefSpeed (10); //Beim Suchen nach der Referenzmarke wird mit 0.1 U/s gefahren.

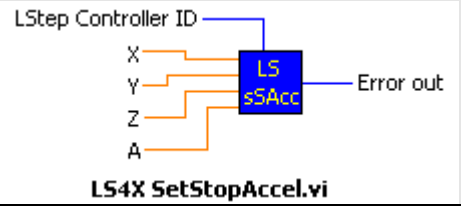
LS_GetRMOffset	
Beschreibung:	RM-Offset abfragen
Delphi:	function LS_GetRMOffset(var X, Y, Z, A: Double): Integer; function LSX_GetRMOffset(LSID: Integer; var X, Y, Z, R: Double): Integer;
C++:	int GetRMOffset (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z und A: RM-Offset, abhängig von Dimension
Beispiel:	LS.GetRMOffset(&X, &Y, &Z, &A);

LS_SetRMOffset	
Beschreibung:	RM-Offset
Delphi:	function LS_SetRMOffset(X, Y, Z, A: Double): Integer; function LSX_SetRMOffset(LSID: Integer; X, Y, Z, R: Double): Integer;
C++:	int SetRMOffset (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, y, z und a 0 – 32*50000 (32*Spindelsteigung)
Beispiel:	LS.SetRMOffset(1, 1, 1, 1); (Die Achsen X, Y und Z werden beim Tischhub messen jeweils 1mm (bei Dim 2 2 2) vom Endendschalter in Richtung Tischmitte verfahren und dann die Softwaregrenze gesetzt.

LS_GetSpeedPoti	
Beschreibung:	Gibt an ob Potentiometer Ein oder Aus ist.
Delphi:	function LS_GetSpeedPoti(var SpePoti: LongBool): Integer; function LSX_GetSpeedPoti(LSID: Integer; var SpePoti: LongBool): Integer;
C++:	int GetSpeedPoti (BOOL *pbSpePoti);
LabView:	
Parameter:	Das SpePoti Flag gibt an, ob Potentiometer Ein oder Aus ist.
Beispiel:	LS.GetSpeedPoti(&flag);

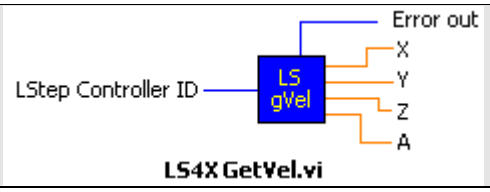
LS_SetSpeedPoti	
Beschreibung:	Potentiometer Ein/Aus
Delphi:	function LS_SetSpeedPoti(SpeedPoti: LongBool): Integer; function LSX_SetSpeedPoti(LSID: Integer; SpeedPoti: LongBool): Integer;
C++:	int SetSpeedPoti (BOOL SpeedPoti);
LabView:	
Parameter:	Bei SpeedPoti = false wird die vorgegebene Geschwindigkeit (vel) als Verfahrgeschwindigkeit genutzt. Bei SpeedPoti = true wird die vorgegebene Geschwindigkeit (vel), in Abhängigkeit von der Stellung des Potentiometers, prozentual genutzt.
Beispiel:	LS.SetSpeedPoti(true); // Poti An

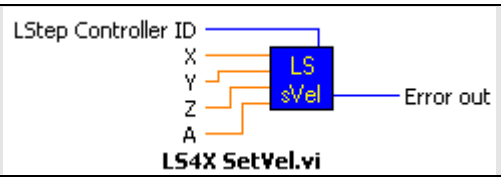
LS_GetStopAccel	
Beschreibung:	Liefert Die Bremsbeschleunigung, wenn der Stopeingang aktiv wird.
Delphi:	function LS_GetStopAccel (var dXD, dYD, dZD, dAD: Double): Integer; function LSX_ GetStopAccel (LSID: Integer; var dXD, dYD, dZD, dAD: Double): Integer;
C++:	int GetStopAccel (double *pdXD, double *pdYD, double *pdZD, double *pdAD);
LabView:	
Parameter:	XD, YD, ZD, AD: Bremsbeschleunigungswerte, [m/s²]
Beispiel:	LS. GetStopAccel (&XD, &YD, &ZD, &AD);

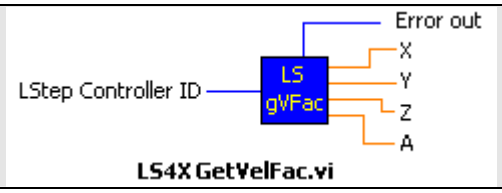
LS_SetStopAccel	
Beschreibung:	Die Bremsbeschleunigung, wenn der Stopeingang aktiv wird, setzen. Mit der Beschleunigung wird bei aktiv werden des Stopeingangs angehalten, außer wenn der Wert der mit LS_SetAccel gesetzten Beschleunigung größer ist. Die angestellte Bremsbeschleunigung gilt nur für den Vektorenbetrieb, nicht für Joystick, Kalibrieren und Hubmessen. Der Wert wird nicht mit LStepSave gespeichert.
Delphi:	function LS_SetStopAccel (dXD, dYD, dZD, dAD: Double): Integer; function LSX_ SetStopAccel (LSID: Integer; dXD, dYD, dZD, dAD: Double): Integer;
C++:	int SetStopAccel (double dXD, double dYD, double dZD, double dAD);
LabView:	
Parameter:	dXD, dYD, dZD, dAD: Bremsbeschleunigung, Wertebereich 0.01 bis 20 m/s²
Beispiel:	LS. SetStopAccel (15.0, 15.0, 15.0, 15.0);

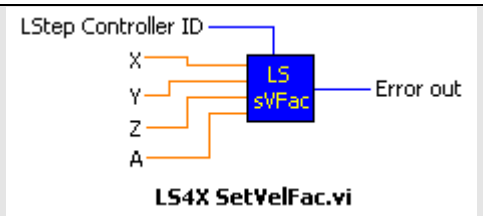
LS_GetStopPolarity	
Beschreibung:	Stopeingang Polarität lesen.
Delphi:	function LS_GetStopPolarity(var bHighActiv: LongBool): Integer; function LSX_ GetStopPolarity (LSID: Integer; var bHighActiv: LongBool): Integer;
C++:	int GetStopPolarity (BOOL *pbHighActiv);
LabView:	
Parameter:	bHighActiv: True – Stopeingang highaktiv False – lowaktiv
Beispiel:	LS. GetStopPolarity (&HighActiv);

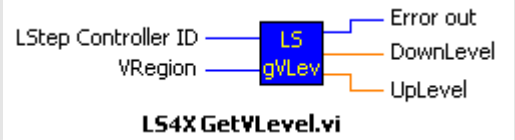
LS_SetStopPolarity	
Beschreibung:	Stopeingang Polarität einstellen. Da der Stopeingang einen Pull Up nach 5V hat, muß man bei einem Schließer lowaktiv und bei einem Öffner highaktiv einstellen.
Delphi:	function LS_SetStopPolarity(bHighActiv: LongBool): Integer; function LSX_ SetStopPolarity (LSID: Integer; bHighActiv: LongBool): Integer;
C++:	int SetStopPolarity (BOOL bHighActiv);
LabView:	
Parameter:	bHighActiv: True – Stopeingang highaktiv False – lowaktiv
Beispiel:	LS. SetStopPolarity (False); //Der Stopeingang ist lowaktiv.

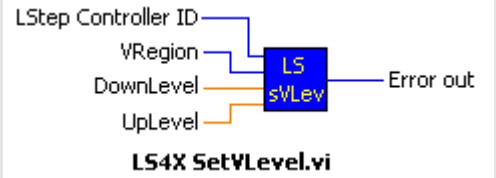
LS_GetVel	
Beschreibung:	Geschwindigkeit abfragen
Delphi:	function LS_GetVel(var X, Y, Z, R: Double): Integer; function LSX_GetVel(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetVel (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Geschwindigkeitswerte [U/s]
Beispiel:	LS.GetVel(&X, &Y, &Z, &A);

LS_SetVel	
Beschreibung:	Geschwindigkeit einstellen
Delphi:	function LS_SetVel(X, Y, Z, R: Double): Integer; function LSX_SetVel(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetVel(double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z und A 0 – maximale Geschwindigkeit [U/s]
Beispiel:	LS.SetVel(1.0, 15.0, 0, 0);

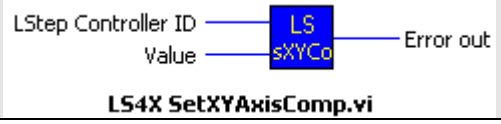
LS_GetVelFac	
Beschreibung:	Geschwindigkeitsuntersetzung abfragen
Delphi:	function LS_ GetVelFac (var X, Y, Z, R: Double): Integer; function LSX_ GetVelFac (LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int SetVelFac (double dX, double dY, double dZ, double dR);
LabView:	
Parameter:	X, Y, Z, A: Geschwindigkeitsuntersetzungswerte
Beispiel:	LS. SetVelFac (&X, &Y, &Z, &A);

LS_SetVelFac	
Beschreibung:	Geschwindigkeitsuntersetzung setzen
Delphi:	function LS_ SetVelFac (X, Y, Z, R: Double): Integer; function LSX_ SetVelFac (LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetVelFac (double dX, double dY, double dZ, double dR);
LabView:	
Parameter:	X, Y, Z, A: Geschwindigkeitsuntersetzung, Wertebereich 0.01 – 1.00
Beispiel:	LS. SetVelFac (0.1, 0.1, 0.1, 0.1); /* reduziert die Geschwindigkeiten aller Achsen auf 1/10 der eingestellten Geschwindigkeiten */

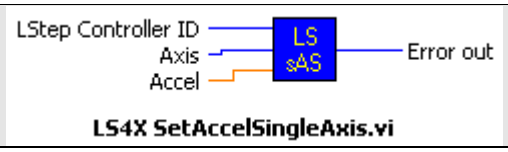
LS_GetVLevel	
Beschreibung:	Liefert die Geschwindigkeitsgrenzen von dem angegebenen Geschwindigkeitsbereich.
Delphi:	function LS_GetVLevel(lVRegion: Integer; var dDownLevel, dUppLevel: Double): Integer; function LSX_ GetVLevel (LSID: Integer; lVRegion: Integer; var dDownLevel, dUppLevel: Double): Integer;
C++:	int GetVLevel (int lVRegion, double *pdDownLevel, double *pdUppLevel);
LabView:	
Parameter:	lVRegion: Wertebereich 1-4. 1 – Erstes/Unteres Geschwindigkeitsbereich 2 – Zweites/Mittleres Geschwindigkeitsbereich 3 – Drittes/Oberes Geschwindigkeitsbereich 4 – Bis zu dieser Geschwindigkeitsgrenze wird die Korrekturtabelle genutzt. dDownLevel : Untere Grenze des Bereichs (bei lVRegion = 4 Geschwindigkeitsgrenze) [U/s] dUppLevel : Obere Grenze des Bereichs (bei lVRegion = 4 hat keine Bedeutung) [U/s]
Beispiel:	LS. GetVLevel (2, &DownLevel, &UppLevel); // DownLevel = Untere Grenze des zweiten Geschwindigkeitsbereich, UppLevel = Obere Grenze des zweiten Geschwindigkeitsbereich.

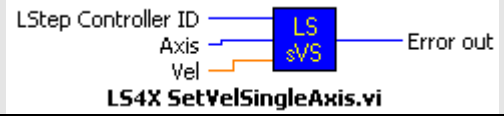
LS_SetVLevel	
Beschreibung:	Geschwindigkeitsbereich ausklammern, in denen das System zu Resonanzen neigt.
Delphi:	function LS_SetVLevel(lVRegion: Integer; dDownLevel, dUppLevel: Double): Integer; function LSX_ SetVLevel (LSID: Integer; lVRegion: Integer; dDownLevel, dUppLevel: Double): Integer;
C++:	int SetVLevel (int lVRegion, double dDownLevel, double dUppLevel);
LabView:	
Parameter:	lVRegion: Wertebereich 1-4. 1 – Erstes/Unteres Geschwindigkeitsbereich 2 – Zweites/Mittleres Geschwindigkeitsbereich 3 – Drittes/Oberes Geschwindigkeitsbereich 4 – Bis zu dieser Geschwindigkeitsgrenze wird die Korrekturtabelle genutzt. dDownLevel : Untere Grenze des Bereichs (bei lVRegion=4 Geschwindigkeitsgrenze) [U/s], Wertebereich 0 – max. Geschwindigkeit. dUppLevel : Obere Grenze des Bereichs (bei lVRegion=4 hat keine Bedeutung) [U/s], Wertebereich 0 – max. Geschwindigkeit.
Beispiel:	LS. SetVLevel (4, 10.0, 0.0); //Die Korrekturtabelle wirkt bis zu einer Geschwindigkeit von 10 Umdrehungen/s.

LS_GetXYAxisComp	
Beschreibung	Abfrage XY-Achsüberlagerung
Delphi	function LS_GetXYAxisComp(var Value: Integer): Integer; function LSX_GetXYAxisComp(LSID: Integer; var Value: Integer): Integer;
C++	int GetXYAxisComp (int *plValue);
LabView:	
Parameter	Value: Modus der Achsüberlagerung (siehe LStep-Dokumentation)
Beispiel	LS.SetXYAxisComp(&mode) ;

LS_SetXYAxisComp	
Beschreibung	XY-Achsüberlagerung aktivieren
Delphi	function LS_SetXYAxisComp(Value: Integer): Integer; function LSX_SetXYAxisComp(LSID: Integer; Value: Integer): Integer;
C++	int SetXYAxisComp (int lValue);
LabView:	
Parameter	Value: Modus der Achsüberlagerung (siehe LStep-Dokumentation)
Beispiel	LS.SetXYAxisComp(1) ;

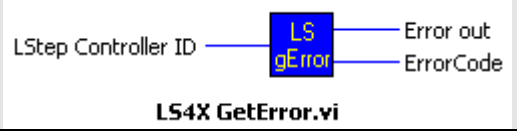
LS_LStepSave	
Beschreibung	Aktuelle Konfiguration in LStep speichern (EEPROM)
Delphi	function LS_LStepSave(): Integer; function LSX_LStepSave(LSID: Integer): Integer;
C++	int LStepSave ();
LabView:	
Parameter	-
Beispiel	LS.LStepSave() ;

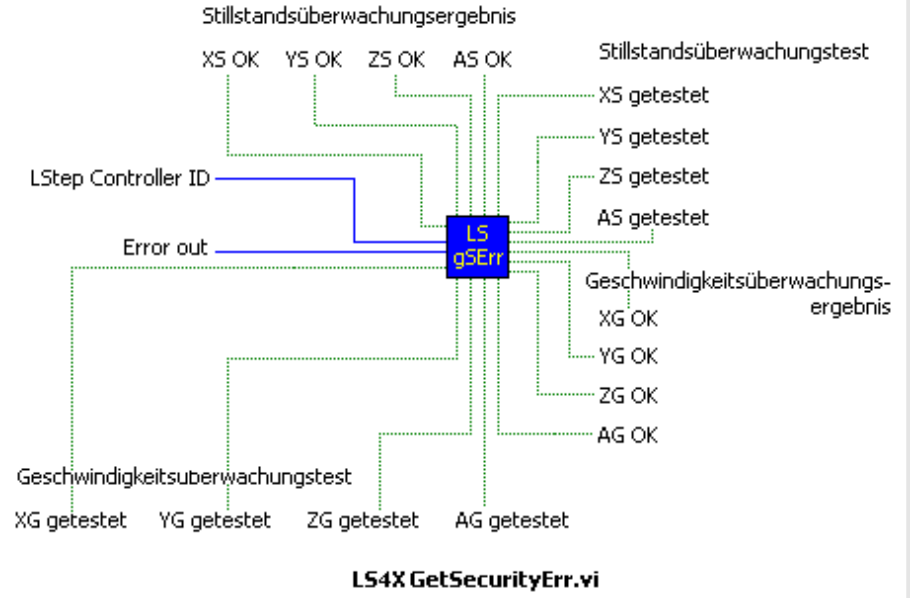
LS_SetAccelSingleAxis	
Beschreibung:	Beschleunigung einstellen
Delphi:	function LS_SetAccelSingleAxis(Axis: Integer; Accel: Double): Integer; function LSX_SetAccelSingleAxis(LSID: Integer; Axis: Integer; Accel: Double): Integer;
C++:	int SetAccelSingleAxis (int lAxis,double dAccel);
LabView:	
Parameter:	Axis: (X, Y, Z, A numeriert von 1 bis 4) Accel: Beschleunigung 0.01 – 10.00 [m/s²]
Beispiel:	LS.SetAccelSingleAxis(4, 1.0); // Beschleunigung A-Achse 1.0 m/s²

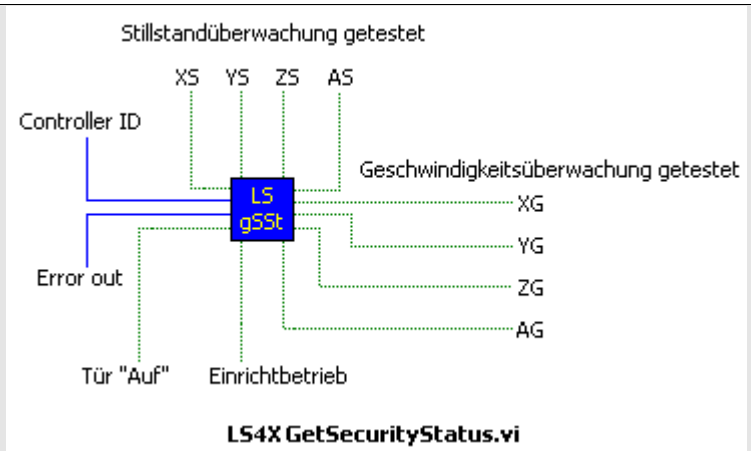
LS_SetVelSingleAxis	
Beschreibung:	Geschwindigkeit für einzelne Achse einstellen
Delphi:	function LS_SetVelSingleAxis(Axis: Integer; Vel: Double): Integer; function LSX_SetVelSingleAxis(LSID: Integer; Axis: Integer; Vel: Double): Integer;
C++:	int SetVelSingleAxis (int lAxis,double dVel);
LabView:	
Parameter:	Axis: (X, Y, Z, A numeriert von 1 bis 4) Vel: 0 – maximale Geschwindigkeit [U/s]
Beispiel:	LS.SetVelSingleAxis(1, 10.0) // Geschwindigkeit X-Achse 10 U/s

LS_SoftwareReset	
Beschreibung:	Die Software wird in den Startzustand versetzt.
Delphi:	function LS_SoftwareReset: Integer; function LSX_SoftwareReset(LSID: Integer): Integer;
C++:	int SoftwareReset ();
LabView:	
Parameter:	-
Beispiel:	LS.SoftwareReset ();

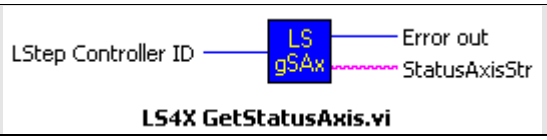
Statusabfragen

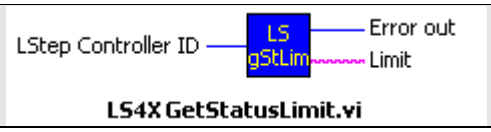
LS_GetError	
Beschreibung:	liefert die aktuelle Fehlernummer
Delphi:	function LS_GetError(var ErrorCode: Integer): Integer; function LSX_GetError(LSID: Integer; var ErrorCode: Integer): Integer;
C++:	int GetError (int *plErrorCode);
LabView:	
Parameter:	ErrorCode: Fehlernummer
Beispiel:	LS.GetError(&ErrCode);

LS_GetSecurityErr	
Beschreibung:	Liest alle Zustände und Ergebnisse der GAL-Sicherheitsüberwachung (nur bei LS44-Steuerungen)
Delphi:	function LS_ Get SecurityErr (var Value: LongWord): Integer; function LSX_ GetS SecurityErr (LSID: Integer; var Value: LongWord): Integer;
C++:	int GetSecurityErr (LongWord *pValue);
LabView:	
Parameter:	Value: 32-Bit LongWord ohne Vorzeichen, welcher nach Aufruf der Funktion in den Bits 0-15 die Bit-Maske enthält. Bit 0 X-Achse Stillstandüberwachungsergebnis (OK [1] / nicht OK [0]) Bit 1 Y-Achse Stillstandüberwachungsergebnis Bit 2 Z-Achse Stillstandüberwachungsergebnis Bit 3 A-Achse Stillstandüberwachungsergebnis Bit 5 X-Achse Stillstandüberwachungstest (getestet [1] /nicht getestet [0]) Bit 6 Y-Achse Stillstandüberwachungstest Bit 7 Z-Achse Stillstandüberwachungstest Bit 8 A-Achse Stillstandüberwachungstest Bit 9 X-Achse Geschwindigkeitsüberwachungsergebnis Bit 10 Y-Achse Geschwindigkeitsüberwachungsergebnis Bit 11 Z-Achse Geschwindigkeitsüberwachungsergebnis Bit 12 A-Achse Geschwindigkeitsüberwachungsergebnis Bit 13 X-Achse Geschwindigkeitsüberwachungstest Bit 14 Y-Achse Geschwindigkeitsüberwachungstest Bit 15 Z-Achse Geschwindigkeitsüberwachungstest
Beispiel:	LS.GetSecurityErr (&Value);

LS_GetSecurityStatus	
Beschreibung:	Liefert den aktuellen Zustand der Sicherheitsüberwachung (nur bei LS44-Steuerungen)
Delphi:	function LS_ Get SecurityStatus (var Value: LongWord): Integer; function LSX_ GetS SecurityStatus (LSID: Integer; var Value: LongWord): Integer;
C++:	int GetSecurityStatus (LongWord *pValue);
LabView:	
Parameter:	Value: 32-Bit LongWord ohne Vorzeichen, welcher nach Aufruf der Funktion in den Bits 0-15 die Bit-Maske enthält. Bit 0-3 interne Merker Bit 4 X-Achse Stillstandüberwachung getestet Bit 5 Y-Achse Stillstandüberwachung getestet Bit 6 Z-Achse Stillstandüberwachung getestet Bit 7 A-Achse Stillstandüberwachung getestet Bit 8 X-Achse Geschwindigkeitsüberwachung getestet Bit 9 Y-Achse Geschwindigkeitsüberwachung getestet Bit 10 Z-Achse Geschwindigkeitsüberwachung getestet Bit 11 A-Achse Geschwindigkeitsüberwachung getestet Bit 14 Zustand Einrichtbetrieb (Einrichtbetrieb = 1) Bit 15 Zustand Tür (Tür „Auf“ = 1)
Beispiel:	LS.GetSecurityStatus (&Value);

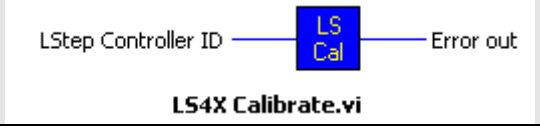
LS_GetStatus	
Beschreibung:	liefert den aktuellen Zustand der Steuerung.
Delphi:	function LS_GetStatus(Stat: PChar; MaxLen: Integer): Integer; function LSX_GetStatus(LSID: Integer; Stat: PChar; MaxLen: Integer): Integer;
C++:	int GetStatus (char *pcStat,int lMaxLen);
LabView:	
Parameter:	Stat: Zeiger auf einen Puffer, in dem der Statusstring zurückgegeben wird MaxLen: Maximale Anzahl von Zeichen, die in den Puffer kopiert werden dürfen
Beispiel:	LS.GetStatus(pcStat, 256);

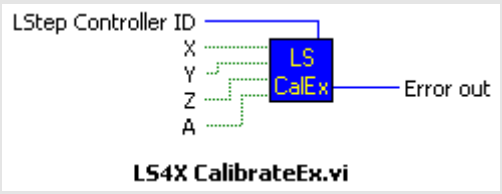
LS_GetStatusAxis	
Beschreibung:	liefert den aktuellen Zustand der einzelnen Achsen
Delphi:	function LS_GetStatusAxis(StatusAxisStr: PChar; MaxLen: Integer): Integer; function LSX_GetStatusAxis(LSID: Integer; StatusAxisStr: PChar; MaxLen: Integer): Integer;
C++:	int GetStatusAxis (char *pcStatusAxisStr,int lMaxLen);
LabView:	
Parameter:	StatusAxisStr: Zeiger auf einen Puffer, in dem der Statusstring zurückgegeben wird MaxLen: Maximale Anzahl von Zeichen, die in den Puffer kopiert werden dürfen z.B.: @ - M – J – C – S – A – D – U - T @ = Achse steht M = Achse ist in Bewegung (Motion) - = Achse ist nicht freigegeben J = Joystick eingeschaltet C = Achse ist in Regelung A = Rückmeldung nach dem Kalibrieren E = Fehler beim Kalibrieren (Endschalter nicht korrekt freigefahren) D = Rückmeldung nach dem Tischhubmessen U = Einrichtbetrieb T = Timeout
Beispiel:	LS.GetStatusAxis(pcStatAxis, 256);

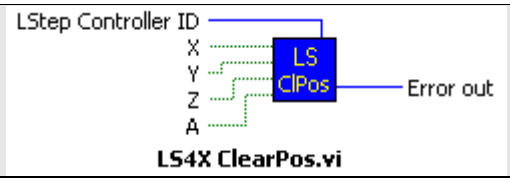
LS_GetStatusLimit	
Beschreibung:	Liefert den aktuellen Zustand der Software-Grenzen jeder einzelnen Achse
Delphi:	function LS_ GetStatusLimit (Limit: PChar; MaxLen: Integer): Integer; function LSX_ GetStatusLimit (LSID: Integer; Limit: PChar; MaxLen: Integer): Integer;
C++:	int GetStatusLimit (char *pcLimit, int lMaxLen);
LabView:	
Parameter:	pc Limit: Zeiger auf einen Puffer, in dem der Zustand der Achsen zurückgegeben wird. Z. B.: AA- A- - DD - LL- L- - L A = Achse wurde kalibriert D = Tischhub wurde gemessen L = Software-Limit wurde gesetzt - = Software-Grenze wurde nicht verändert MaxLen: Maximale Anzahl von Zeichen, die in den Puffer kopiert werden dürfen
Beispiel:	LS.GetStatusLimit (pc Limit, 64);

LS_SetAutoStatus	
Beschreibung:	AutoStatus Ein/Aus
	Hinweis: der AutoStatus-Modus sollte normalerweise nicht verändert werden, da das LSTEP API bei Verfahrbefehlen etc. den richtigen Modus einstellt, eine Änderung auf 0 oder 2 könnte zu Fehlern führen
Delphi:	function LS_SetAutoStatus(Value: Integer): Integer; function LSX_SetAutoStatus(LSID: Integer; Value: Integer): Integer;
C++:	int SetAutoStatus (int lValue);
LabView:	
Parameter:	Value: AutoStatus-Modus:
	0 ➔ Es wird kein Status von der Steuerung gesendet. 1 ➔ Es werden automatisch „Positionerreicht“- Meldungen von der Steuerung gesendet. 2 ➔ Es werden automatisch „Positionerreicht“- und Status - Meldungen von der Steuerung gesendet. 3 ➔ Es gibt bei „Positionerreicht“ nur ein Carriage Return zurück.
Beispiel:	LS.SetAutoStatus(3);

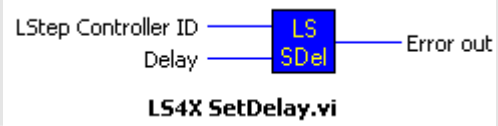
Fahrbefehle und Positionsverswaltung

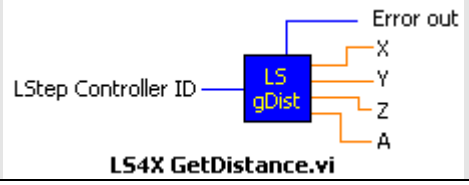
LS_Calibrate	
Beschreibung:	Kalibrieren
	Bewegt alle freigegebenen Achsen in Richtung kleinerer Positionswerte. Die Verfahrbewegung wird unterbrochen sobald die Endschalter angefahren wurden. Der Positionswert wird auf 0 gesetzt.
Delphi:	function LS_Calibrate: Integer; function LSX_Calibrate(LSID: Integer): Integer;
C++:	int Calibrate();
LabView:	
Parameter:	-
Beispiel:	LS.Calibrate();

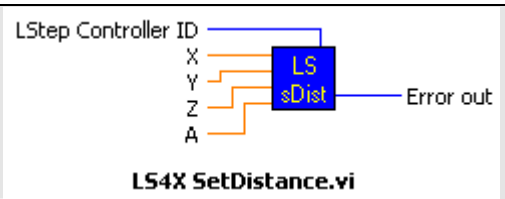
LS_CalibrateEx	
Beschreibung:	Kalibrieren
	(Es werden nur die Achsen kalibriert, deren entsprechendes Bit in dem übergebenen Integer-Wert gesetzt ist.)
Delphi:	function LS_CalibrateEx(Flags: Integer): Integer; function LSX_CalibrateEx(LSID: Integer; Flags: Integer): Integer;
C++:	int CalibrateEx (int lFlags);
LabView:	
Parameter:	Flags: Bit-Maske, Bit 2 = 1 → Z-Achse kalibrieren Bit 2 = 0 → Z-Achse nicht kalibrieren ...
Beispiel:	LS.CalibrateEx(6); // Nur Y- und Z-Achse kalibrieren

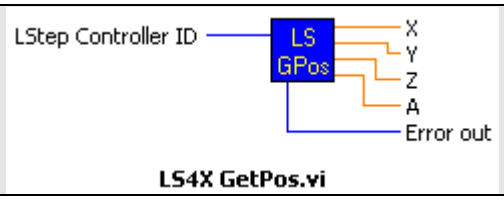
LS_ClearPos	
Beschreibung:	Setzt die Position auf 0, auch den internen Zähler. Diese Funktion wird für Endlosachsen gebraucht, da die Steuerung nur ±1000 Motorumdrehungen vom Werteberreich verarbeiten kann. Bei erkannten Geber wird die Funktion für jeweilige Achse nicht ausgeführt.
Delphi:	function LS_ClearPos (lFlags: Integer): Integer; function LSX_ClearPos (LSID: Integer; lFlags: Integer): Integer;
C++:	int ClearPos (int lFlags);
LabView:	
Parameter:	lFlags: Bit-Maske Bit 0 = 1 → Position der x-Achse wird genullt Bit 1 = 0 → Für die y-Achse wird die Function nicht ausgeführt
Beispiel:	LS. ClearPos(5); //Positionen der x- und z- Achsen werden genullt.

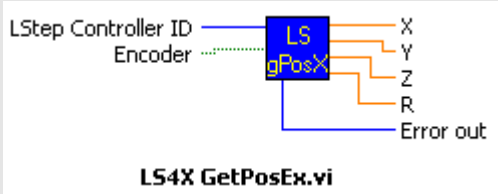
LS_GetDelay	
Beschreibung:	Liest die Verzögerung des Vektorstarts.
Delphi:	function LS_GetDelay(var Delay: Integer): Integer; function LSX_GetDelay(LSID: Integer; var Delay: Integer): Integer;
C++:	int GetDelay (int *plDelay);
LabView:	
Parameter:	Delay: Verzögerung, in ms
Beispiel:	LS.GetDelay(&Delay);

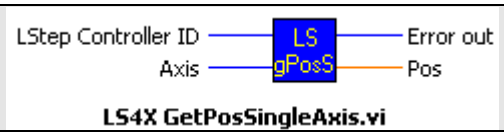
LS_SetDelay	
Beschreibung:	Durch den Befehl Delay kann eine Verzögerung des Vektorstarts erzeugt werden.
Delphi:	function LS_SetDelay(Delay: Integer): Integer; function LSX_SetDelay(LSID: Integer; Delay: Integer): Integer;
C++:	int SetDelay (int lDelay);
LabView:	
Parameter:	0 – 10000 (ms)
Beispiel:	LS.SetDelay(1000); // 1s Verzögerung

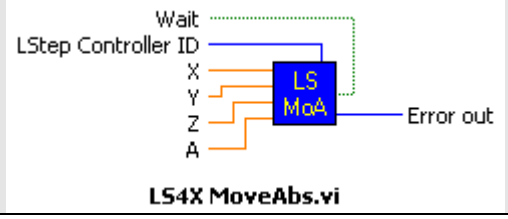
LS_GetDistance	
Beschreibung:	Liefert die Strecke für LS_MoveRelShort
Delphi:	function LS_GetDistance(var X, Y, Z, A: Double): Integer; function LSX_GetDistance(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetDistance (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z und A: die aktuellen Strecken aller Achsen, abhängig von den Dimensionen
Beispiel:	LS.GetDistance(&X, &Y, &Z, &A);

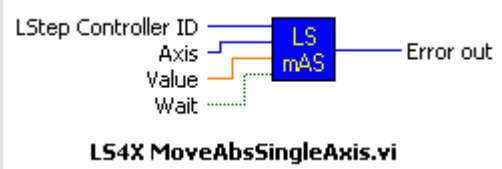
LS_SetDistance	
Beschreibung:	Strecke setzen (für LS_MoveRelShort)
Delphi:	function LS_SetDistance(X, Y, Z, A: Double): Integer; function LSX_SetDistance(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetDistance (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z und A Min-/max-Verfahrbereich (Werte sind abhängig von der Dimension)
Beispiel:	LS.SetDistance(1, 2, 0, 0); /* Strecken für die Achsen X und Y werden gesetzt, Z und A werden bei Aufruf der Funktion LS_MoveRelShort nicht bewegt. */

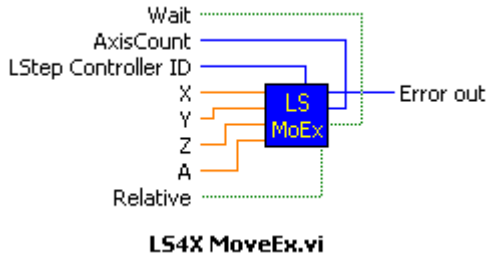
LS_GetPos	
Beschreibung:	Abfrage der aktuellen Position aller Achsen
	Für nicht vorhandene Achsen wird der Wert 0.0 zurückgeliefert
Delphi:	function LS_GetPos(var X, Y, Z, A: Double): Integer; function LSX_GetPos(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetPos (double *pdX,double *pdY,double *pdZ,double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Positionswerte
Beispiel:	double X, Y, Z, A; LS.GetPos(&X, &Y, &Z, &A);

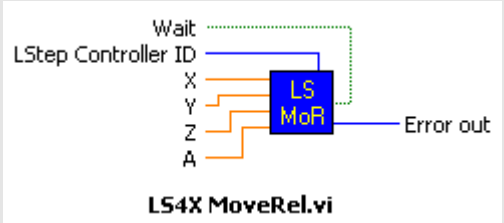
LS_GetPosEx	
Beschreibung:	Abfrage der aktuellen Geber- bzw. Positionswerte aller Achsen
	Für nicht vorhandene Achsen wird der Wert 0.0 zurückgeliefert
Delphi:	function LS_GetPosEx(var X, Y, Z, A: Double; Encoder: LongBool): Integer; function LSX_GetPosEx(LSID: Integer; var X, Y, Z, R: Double; Encoder: LongBool): Integer;
C++:	int GetPosEx (double *pdX,double *pdY,double *pdZ,double *pdA,BOOL Encoder);
LabView:	
Parameter:	X, Y, Z, A: Positionswerte Encoder = true → Geberwerte liefern falls Geber angeschlossen Encoder = false → Positionswerte liefern
Beispiel:	double X, Y, Z, A; LS.GetPosEx(&X, &Y, &Z, &A, true);

LS_GetPosSingleAxis	
Beschreibung:	Abfrage der aktuellen Position einer Achse
	Für nicht vorhandene Achsen wird der Wert 0.0 zurückgeliefert
Delphi:	function LS_GetPosSingleAxis(Axis: Integer; var Pos: Double): Integer; function LSX_GetPosSingleAxis(LSID: Integer; Axis: Integer; var Pos: Double): Integer;
C++:	int GetPosSingleAxis (int lAxis,double *pdPos);
LabView:	
Parameter:	Axis: Achse, deren Positionswert abgefragt werden soll (X, Y, Z, A numeriert von 1 bis 4) Pos: Positionswert
Beispiel:	LS.GetPosSingleAxis(2, &YPos); // Position Y-Achse auslesen

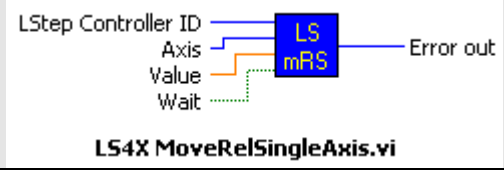
LS_MoveAbs	
Beschreibung:	Absolutposition anfahren
	(Die Achsen x, y, z und a werden auf die übergebenen Positionswerte positioniert.)
Delphi:	function LS_MoveAbs(X, Y, Z, A: Double; Wait: LongBool): Integer; function LSX_MoveAbs(LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;
C++:	int MoveAbs (double dX, double dY, double dZ, double dA, BOOL Wait);
LabView:	
Parameter:	X, Y, Z und A +- Verfahrbereich Eingabe ist abhängig von der Dimension Wait: Gibt an, ob die Funktion nachdem die Position erreicht wurde (= true) oder direkt zurückkehren soll (= false)
Beispiel:	LS.MoveAbs(10.0, 10.0, 10.0, 10.0, true);

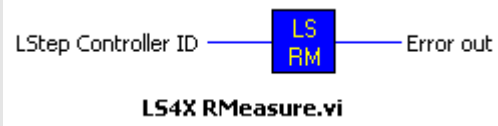
LS_MoveAbsSingleAxis	
Beschreibung:	Einzelne Achse absolut positionieren
Delphi:	function LS_MoveAbsSingleAxis(Axis: Integer; Value: Double; Wait: LongBool): Integer; function LSX_MoveAbsSingleAxis(LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;
C++:	int MoveAbsSingleAxis (int lAxis,double dValue,BOOL Wait);
LabView:	
Parameter:	Axis: (X, Y, Z, A numeriert von 1 bis 4) Value: Position (Eingabe ist abhängig von der eingestellten Dimension)
Beispiel:	LS.MoveAbsSingleAxis(2, 10.0); // Y-Achse auf 10mm absolut positionieren

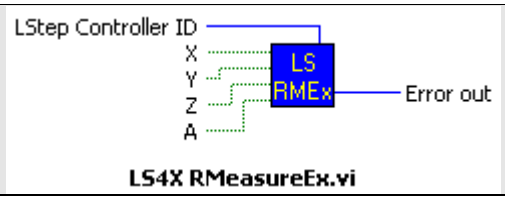
LS_MoveEx	
Beschreibung	Erweiterter Verfahr-Befehl Die Funktion LS_MoveEx kann relative und absolute Verfahrbefehle ausführen, synchron und asynchron. Die Anzahl der Achsen, die verfahren werden sollen, kann mit dem Parameter AxisCount bestimmt werden Diese Funktion kann beispielsweise genutzt werden, um bei einer LStep44 nur X und Y zu verfahren.
Delphi	function LS_MoveEx(X, Y, Z, R: Double; Relative, Wait: LongBool; AxisCount: Integer): Integer; function LSX_MoveEx(LSID: Integer; X, Y, Z, R: Double; Relative, Wait: LongBool; AxisCount: Integer): Integer;
C++	int MoveEx (double dX, double dY, double dZ, double dR, BOOL bRelative, BOOL bWait, int lAxisCount);
LabView:	
Parameter	X, Y, Z, R : Positions-Vektor Relative : bei Relative=false werden die Werte X, Y, Z und R als absolute Koordinaten interpretiert, bei Relative=true als relative Koordinaten zur aktuellen Position Wait: wird Wait=true gesetzt, kehrt die Funktion erst nach Erreichen der Zielposition zurück, ansonsten kehrt sie unmittelbar nach Senden des Befehls an die LStep zurück. AxisCount: Anzahl der Achsen, die verfahren werden sollen Ist AxisCount=1, wird nur X verfahren Ist AxisCount=2, werden X und Y verfahren...
Beispiel	LS_MoveEx(2.0, 3.0, 0, 0, true, true, 2) ; // Es werden X und Y um 2 bzw 3 relativ verfahren

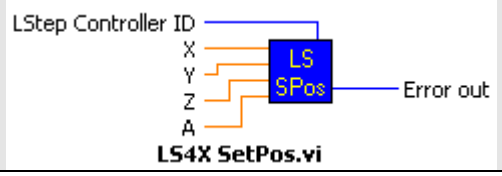
LS_MoveRel	
Beschreibung:	Relativen Vektor fahren
	(Die Achsen x, y, z und a werden um die übergebenen Strecken verfahren.)
Delphi:	function LS_MoveRel(X, Y, Z, A: Double; Wait: LongBool): Integer; function LSX_MoveRel(LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;
C++:	int MoveRel (double dX, double dY, double dZ, double dA, BOOL Wait);
LabView:	
Parameter:	X, Y, Z und A +- Verfahrbereich Eingabe ist abhängig von der Dimension Wait: Gibt an, ob die Funktion nachdem die Position erreicht wurde (= true) oder direkt zurückkehren soll (= false)
Beispiel:	LS.MoveRel(10.0, 10.0, 10.0, 10.0, true);

LS_MoveRelShort	
Beschreibung:	Positionieren Relativ (short command)
	Dieser Befehl sollte verwendet werden, damit aufeinander folgende relative Verfahrbefehle (mit derselben Strecke) schneller angefahren werden. Die Strecke muß zuvor einmal mit LS_SetDistance gesetzt worden sein.
Delphi:	function LS_MoveRelShort: Integer; function LSX_MoveRelShort(LSID: Integer): Integer;
C++:	int MoveRelShort ();
LabView:	
Parameter:	-
Beispiel:	LS.SetDistance(1.0, 1.0, 0, 0); for (i = 0; i < 10; i++) LS.MoveRelShort(); // 10mal X- und Y-Achse um 1 mm relativ positionieren

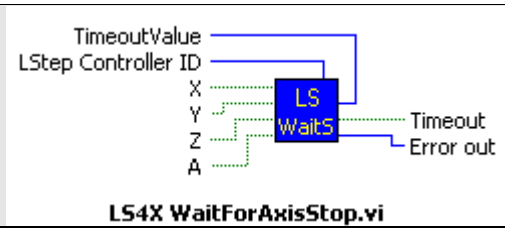
LS_MoveRelSingleAxis	
Beschreibung:	Einzelne Achse relativ verfahren
Delphi:	function LS_MoveRelSingleAxis(Axis: Integer; Value: Double; Wait: LongBool): Integer; function LSX_MoveRelSingleAxis(LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;
C++:	int MoveRelSingleAxis (int lAxis,double dValue,BOOL Wait);
LabView:	
Parameter:	Axis: (X, Y, Z, A numeriert von 1 bis 4) Value: Strecke (Eingabe ist abhängig von der eingestellten Dimension)
Beispiel:	LS.MoveRelSingleAxis(3, 5.0); // Z-Achse um 5mm in positiver Richtung verfahren

LS_RMeasure	
Beschreibung:	Tischhub messen
	Bewegt alle freigegebenen Achsen in Richtung größerer Positionswerte. Die Verfahrbewegung wird unterbrochen sobald die Endschalter angefahren wurden. Der Positionswert wird gespeichert.
Delphi:	function LS_RMeasure: Integer; function LSX_RMeasure(LSID: Integer): Integer;
C++:	int RMeasure();
LabView:	
Parameter:	-
Beispiel:	LS.RMeasure();

LS_RMeasureEx	
Beschreibung:	Tischhub messen (Tischhub messen wird nur bei den Achsen durchgeführt, deren entsprechendes Bit in dem übergebenen Integer-Wert gesetzt ist.
Delphi:	function LS_RMeasureEx(Flags: Integer): Integer; function LSX_RMeasureEx(LSID: Integer; Flags: Integer): Integer;
C++:	int RMeasureEx (int lFlags);
LabView:	
Parameter:	Flags: Bit-Maske, Bit 2 = 1 → Z-Achse kalibrieren Bit 2 = 0 → Z-Achse nicht kalibrieren ...
Beispiel:	LS.RMeasureEx(2); // Tischhub messen (nur Y-Achse)

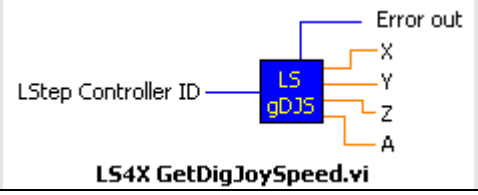
LS_SetPos	
Beschreibung:	Positionswerte setzen
Delphi:	function LS_SetPos(X, Y, Z, R: Double): Integer; function LSX_SetPos(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetPos(double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z und A Min-/Max-Verfahrbereich Eingabe ist abhängig von der Dimension
Beispiel:	LS.SetPos(10, 10, 0, 0);

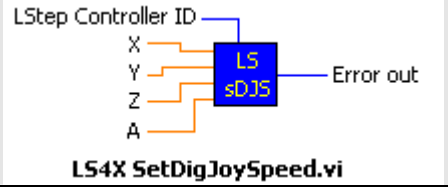
LS_StopAxes	
Beschreibung	Abbruch (Es werden alle Verfahrbewegungen abgebrochen)
Delphi	function LS_StopAxes: Integer; function LSX_StopAxes(LSID: Integer): Integer;
C++	int StopAxes ();
LabView	
Parameter	-
Beispiel	LS.StopAxes();

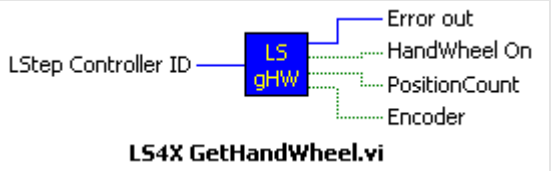
LS_WaitForAxisStop	
Beschreibung	Die Funktion kehrt zurück, sobald die in der Bit-Maske AFlags gewählten Achsen ihre Zielposition erreicht haben LS_WaitForAxisStop verwendet ‚?statusaxis', um den Status der Achsen zu pollen.
Delphi	function LS_WaitForAxisStop(AFlags: Integer; ATimeoutValue: Integer; var ATimeout: LongBool): Integer; function LSX_WaitForAxisStop(LSID: Integer; AFlags: Integer; ATimeoutValue: Integer; var ATimeout: LongBool): Integer;
C++	int WaitForAxisStop (int lAFlags, int lATimeoutValue, BOOL *pbATimeout);
LabView:	
Parameter	AFlags: Bit-Maske Bit 0: X-Achse Bit 1: Y-Achse Bit 2: Z-Achse Bit 3: A-Achse AtimeoutValue: Timeout in Millisekunden, WaitForAxisStop kehrt nach dieser Zeit mit Atimeout=true zurück, wenn die Achsen immer noch in Bewegung sind AtimeoutValue = 0 setzt den Timeout auf 'Unendlich' Das Atimeout Flag gibt an, ob ein Timeout aufgetreten ist.

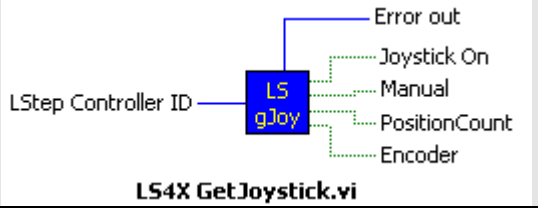
Beispiel	LS.WaitForAxisStop(3, 0, flag);
	// Warten bis X und Y-Achse gestoppt haben, kein Timeout
	LS.WaitForAxisStop(7, 10000, flag);
	// Warten bis X und Y-Achse gestoppt haben, 10 Sekunden timeout

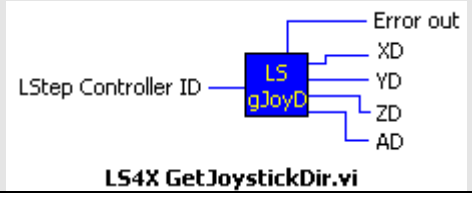
Joystick und Handrad

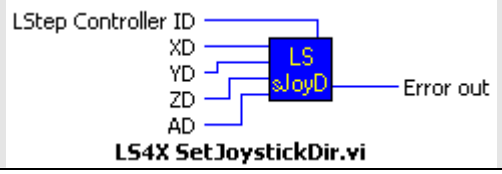
LS_GetDigJoySpeed	
Beschreibung:	Auslesen der eingestellten Geschwindigkeiten
Delphi:	function LS_GetDigJoySpeed(var dX, dY, dZ, dR: Double): Integer; function LSX_GetDigJoySpeed(LSID: Integer; var dX, dY, dZ, dR: Double): Integer;
C++:	int GetDigJoySpeed (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	dX, dY, dZ, dR: Geschwindigkeitswerte [U/s]
Beispiel:	LS. GetDigJoySpeed(&X, &Y, &Z, &R);

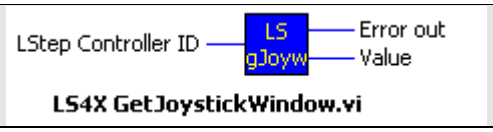
LS_SetDigJoySpeed	
Beschreibung:	Mit diesem Befehl können einzelne Achsen mit einer konstanten Geschwindigkeit verfahren werden. Will man nach dem Ausführen der Funktion wieder absolut oder relativ positionieren, muss man die Geschwindigkeit neu setzen.
Delphi:	function LS_SetDigJoySpeed(dX, dY, dZ, dR: Double): Integer; function LSX_SetDigJoySpeed(LSID: Integer; dX, dY, dZ, dR: Double): Integer;
C++:	int SetDigJoySpeed (double dX, double dY, double dZ, double dR);
LabView:	
Parameter:	dX, dY, dZ, dR: Geschwindigkeit [U/s], Wertebereich: +- max. Geschwindigkeit
Beispiel:	LS. SetDigJoySpeed(0, 10.0, 25.0, 0); //Achsen X und R – Geschwindigkeit 0 und Joystick-Betrieb „AUS“, Achse Y – Geschwindigkeit 10.0 U/s und Joystick-Betrieb „EIN“, Achse Z – Geschwindigkeit 25.0 U/s und Joystick-Betrieb „EIN“.

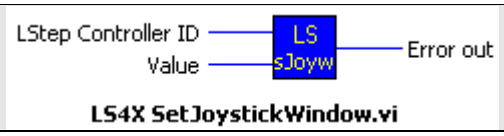
LS_GetHandWheel	
Beschreibung:	Handradszustand ablesen
Delphi:	function LS_GetHandWheel(var PositionCount, Encoder: Boolean): Integer; function LSX_GetHandWheel(LSID: Integer; var PositionCount, Encoder: LongBool): Integer;
C++:	int GetHandWheel (BOOL *pbHandWheelOn, BOOL *pbPositionCount, BOOL *pbEncoder);
LabView:	
Parameter:	HWOn: True = Handrad ist eingeschaltet PosCount: True = Positionszählung ist eingeschaltet Encoder: True = Geberwerte, wenn vorhanden
Beispiel:	LS. GetHandWheel (&HWOn, &PosCount, &Encoder);

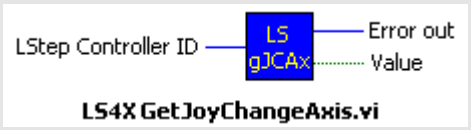
LS_GetJoystick	
Beschreibung:	Abfrage des aktuellen Zustands vom Analog-Joystick.
Delphi:	function LS_GetJoystick (var JoystickOn, Manual, PositionCount, Encoder: LongBool): Integer; function LSX_ GetJoystick (LSID: Integer; var JoystickOn, Manual, PositionCount, Encoder: LongBool): Integer;
C++:	int GetJoystick (BOOL *pbJoystickOn, BOOL *pbManual, BOOL *pbPositionCount, BOOL *pbEncoder);
LabView:	
Parameter:	JoyOn: True = Joystick ist eingeschaltet Manual: False = Joystickschalter steht auf Automatik True = Joystick ist manual über Schalter eingeschaltet PosCount: True = Positionszählung ist eingeschaltet Enc: True = Geberwerte, wenn vorhanden
Beispiel:	LS. GetJoystick (&JoyOn, &Manual, &PosCount, &Enc);

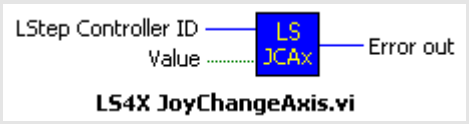
LS_GetJoystickDir	
Beschreibung:	Liest Motordrehrichtung für Joystick
Delphi:	function LS_GetJoystickDir(var XD, YD, ZD, AD: Integer): Integer; function LSX_GetJoystickDir(LSID: Integer; var XD, YD, ZD, AD: Integer): Integer;
C++:	int GetJoystickDir (int *plXD, int *plYD, int *plZD, int *plRD);
LabView:	
Parameter:	X, Y, Z, und A
	0 ➔ Achse gesperrt 1 ➔ positive Drehrichtung -1 ➔ negative Drehrichtung 2 ➔ mit Stromreduzierung -2 ➔ mit Stromreduzierung
Beispiel:	LS.GetJoystickDir(&X, &Y, &Z, &A);

LS_SetJoystickDir	
Beschreibung:	Richtung Joystick
Delphi:	function LS_SetJoystickDir(XD, YD, ZD, AD: Integer): Integer; function LSX_SetJoystickDir(LSID: Integer; XD, YD, ZD, AD: Integer): Integer;
C++:	int SetJoystickDir (int lXD,int lYD,int lZD,int lAD);
LabView:	
Parameter:	X, Y, Z, und A
	0 ➔ Achse gesperrt 1 ➔ positive Drehrichtung -1 ➔ negative Drehrichtung 2 ➔ mit Stromreduzierung -2 ➔ mit Stromreduzierung
Beispiel:	LS.SetJoystickDir(1, 1, -1, 0); /* X- und Y-Achse positive Drehrichtung; Z-Achse negative Drehrichtung; A-Achse gesperrt */

LS_GetJoystickWindow	
Beschreibung	Joystick-Fenster ablesen
Delphi	function LS_GetJoystickWindow(var AValue: Integer): Integer; function LSX_GetJoystickWindow(LSID: Integer; var AValue: Integer): Integer;
C++	int GetJoystickWindow (int *plAValue);
LabView:	
Parameter	AValue: der Analogbereich, in dem sich die Achsen nicht bewegen.
Beispiel	LS.GetJoystickWindow(&AValue) ;

LS_SetJoystickWindow	
Beschreibung	Joystick-Fenster setzen
Delphi	function LS_SetJoystickWindow(AValue: Integer): Integer; function LSX_SetJoystickWindow(LSID: Integer; AValue: Integer): Integer;
C++	int SetJoystickWindow (int lAValue);
LabView:	
Parameter	AValue: 0-100
Beispiel	LS.SetJoystickWindow(20) ;

LS_GetJoyChangeAxis	
Beschreibung:	Liest Joystick-Achszuordnung
Delphi:	LS_GetJoyChangeAxis(var Value: LongBool): Integer; LSX_GetJoyChangeAxis(LSID: Integer; var Value: LongBool): Integer;
C++:	int GetJoyChangeAxis (BOOL *pbValue);
LabView:	
Parameter:	Value: true => Konventionelle Joystickauswertung false => X- und Y-Achszuordnung ist getauscht
Beispiel:	LS. GetJoyChangeAxis (&Value);

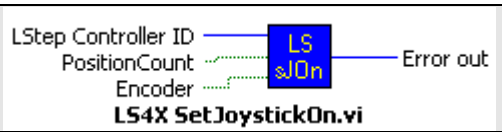
LS_JoyChangeAxis	
Beschreibung:	setzt Joystick-Achszuordnung
Delphi:	LS_JoyChangeAxis(Value: LongBool): Integer; LSX_JoyChangeAxis(LSID: Integer; Value: LongBool): Integer;
C++:	int JoyChangeAxis (BOOL bValue);
LabView:	
Parameter:	Value: 0 - Ändert die Zuordnung der AD-Joystickkanäle (konventionelle Joystickauswertung) 1 - X- und Y-Achszuordnung wird getauscht
Beispiel:	LS. JoyChangeAxis (true);

LS_SetDigJoyOff	
Beschreibung:	Schaltet digitalen Joystick aus
Delphi:	function LS_ SetDigJoyOff: Integer; function LSX_ SetDigJoyOff (LSID: Integer): Integer;
C++:	int SetDigJoyOff () ;
LabView:	
Parameter:	
Beispiel:	LS. SetDigJoyOff () ;

LS_SetHandWheelOff	
Beschreibung:	Handrad Aus
Delphi:	function LS_SetHandWheelOff: Integer; function LSX_SetHandWheelOff(LSID: Integer): Integer;
C++:	int SetHandWheelOff ();
LabView:	
Parameter:	-
Beispiel:	LS.SetHandWheelOff();

LS_SetHandWheelOn	
Beschreibung:	Handrad Ein
Delphi:	function LS_SetHandWheelOn(PositionCount, Encoder: Boolean): Integer; function LSX_SetHandWheelOn(LSID: Integer; PositionCount, Encoder: LongBool): Integer;
C++:	int SetHandWheelOn (BOOL fPositionCount,BOOL fEncoder);
LabView:	
Parameter:	PositionCount: Positionszählung Ein/Aus Encoder: Geberwerte, wenn vorhanden
Beispiel:	LS. SetHandWheelOn (true, true); // Handrad Ein mit Positionszählung (Geberwerte)

LS_SetJoystickOff	
Beschreibung:	Analog-Joystick Aus
Delphi:	function LS_SetJoystickOff: Integer; function LSX_SetJoystickOff(LSID: Integer): Integer;
C++:	int SetJoystickOff();
LabView:	
Parameter:	-
Beispiel:	LS.SetJoystickOff();

LS_SetJoystickOn	
Beschreibung:	Analog-Joystick Ein
Delphi:	function LS_SetJoystickOn(PositionCount, Encoder: LongBool): Integer; function LSX_SetJoystickOn(LSID: Integer; PositionCount, Encoder: LongBool): Integer;
C++:	int SetJoystickOn (BOOL PositionCount,BOOL Encoder);
LabView:	
Parameter:	PositionCount: Positionszählung Ein/Aus Encoder: Geberwerte, wenn vorhanden
Beispiel:	LS.SetJoystickOn(true, true); // Joystick Ein mit Positionszählung (Geberwerte)

Bedienpult mit Trackball und Joyspeed-Tasten

LS_GetBPZ	
Beschreibung:	Liest den Zustand des Zusatzbedienpults mit Trackball
Delphi:	function LS_GetBPZ(var AValue: Integer): Integer; function LSX_GetBPZ(LSID: Integer; var AValue: Integer): Integer;
C++:	int GetBPZ (int *plAValue);
LabView:	-
Parameter:	AValue: 0 => Bedienpult ist „AUS“. 1 => Bedienpult aktiv, Trackball wird mit 0,1µ Schrittauflösung betrieben. 2 => Bedienpult aktiv, Trackball wird mit Faktor betrieben.
Beispiel:	LS.GetBPZ(&AValue);

LS_SetBPZ	
Beschreibung	Bedienpult Ein/Aus
Delphi	function LS_SetBPZ(AValue: Integer): Integer; function LSX_SetBPZ(LSID: Integer; AValue: Integer): Integer;
C++	int SetBPZ (int lAValue);
LabView:	-
Parameter	AValue: 0-2 0 => Bedienpult „AUS“ 1 => Bedienpult aktivieren und Trackball mit 0,1µ Schrittauflösung betreiben. 2 => Bedienpult aktivieren und Trackball mit Faktor betreiben.
Beispiel	LS.SetBPZ(1);

LS_GetBPZJoyspeed	
Beschreibung:	Bedienpult Joystick-Speed
Delphi:	function LS_GetBPZJoyspeed(APar: Integer; var AValue: Double): Integer; function LSX_GetBPZJoyspeed(LSID: Integer; APar: Integer; var AValue: Double): Integer;
C++:	int GetBPZJoyspeed (int lAPar, double *pdAValue);
LabView:	-
Parameter:	APar: 1, 2 oder 3 AValue: maximale Geschwindigkeit [U/s]
Beispiel:	GetBPZJoyspeed(1, &AValue); // Auslesen der eingestellten Geschwindigkeit von Parameter 1.

LS_SetBPZJoyspeed	
Beschreibung	Bedienpult Joystick-Speed
Delphi	function LS_SetBPZJoyspeed(APar: Integer; AValue: Double): Integer; function LSX_SetBPZJoyspeed(LSID: Integer; APar: Integer; AValue: Double): Integer;
C++	int SetBPZJoyspeed (int lAPar, double dAValue);
LabView:	-
Parameter	APar: 1, 2 oder 3 AValue: +- maximale Geschwindigkeit (vel)
Beispiel	SetBPZJoyspeed(1, 25) // Parameter 1 mit Geschwindigkeit 25 beschreiben

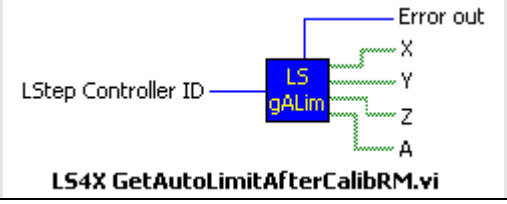
LS_GetBPZTrackballBacklash	
Beschreibung	Bedienpult Trackball-Umkehrspiel auslesen
Delphi	function LS_GetBPZTrackballBackLash(var X, Y, Z, R: Double): Integer; function LSX_GetBPZTrackballBackLash(LSID: Integer; var X, Y, Z, R: Double): Integer;
C++	int GetBPZTrackballBackLash (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	-
Parameter	X, Y, Z, R: Umkehrspiel, mm.
Beispiel	LS.GetBPZTrackballBackLash(&X, &Y, &Z, &R);

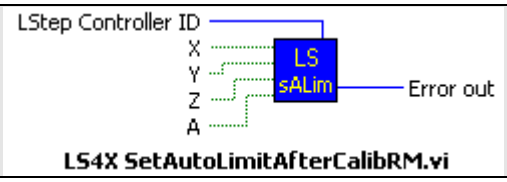
LS_SetBPZTrackballBacklash	
Beschreibung	Bedienpult Trackball-Umkehrspiel
Delphi	function LS_SetBPZTrackballBackLash(X, Y, Z, R: Double): Integer; function LSX_SetBPZTrackballBackLash(LSID: Integer; X, Y, Z, R: Double): Integer;
C++	int SetBPZTrackballBackLash (double dX, double dY, double dZ, double dR);
LabView:	-
Parameter	X, Y, Z, R: 0.001 bis 0.15 mm
Beispiel	LS.SetBPZTrackballBackLash(0.01, 0.01, 0.01, 0.01);

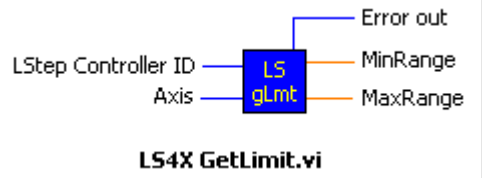
LS_GetBPZTrackballFactor	
Beschreibung	Bedienpult Trackball-Faktor auslesen
Delphi	function LS_GetBPZTrackballFactor(AValue: Double): Integer; function LSX_GetBPZTrackballFactor(LSID: Integer; AValue: Double): Integer;
C++	int GetBPZTrackballFactor (double *pdAValue);
LabView:	-
Parameter	AValue: Trackball – Factor. Z. B. AValue = 3 heißt: Ein Trackball-Impuls ergibt 3 Motor-Incremente.
Beispiel	LS.GetBPZTrackballFactor(&AValue) ;

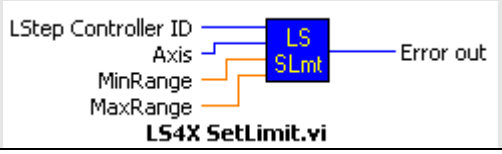
LS_SetBPZTrackballFactor	
Beschreibung	Bedienpult Trackball-Faktor
Delphi	function LS_SetBPZTrackballFactor(AValue: Double): Integer; function LSX_SetBPZTrackballFactor(LSID: Integer; AValue: Double): Integer;
C++	int SetBPZTrackballFactor (double dAValue);
LabView:	-
Parameter	AValue: 0.01 – 100 AValue=1 => Trackball – Factor = 1, d.h. Ein Trackball-Impuls ergibt ein Motor-Increment.
Beispiel	LS.SetBPZTrackballFactor(1.0) ;

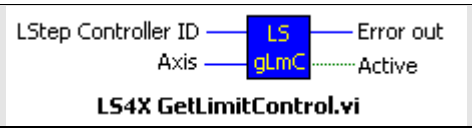
Endschalter (Hardware u. Software)

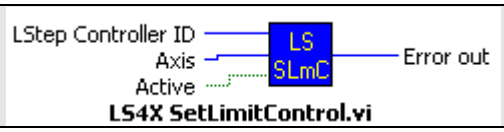
LS_GetAutoLimitAfterCalibRM	
Beschreibung:	Gibt an, ob beim Kalibrieren und Tischhubmessendie interne Software-Limits gesetzt werden.
Delphi:	function LS_GetAutoLimitAfterCalibRM(var lFlags: Integer): Integer; function LSX_ GetAutoLimitAfterCalibRM(LSID: Integer; var lFlags: Integer): Integer;
C++:	int GetAutoLimitAfterCalibRM (int *plFlags);
LabView:	
Parameter:	lFlags: Bit-Maske Bit 0 = 1 → Bei der x-Achse werden keine Verfahrbereichsgrenzen gesetzt Bit 1 = 0 → Für die y-Achse werden Software-Limits gesetzt (calib/rm)
Beispiel:	LS. SetAutoLimitAfterCalibRM(&lFlags);

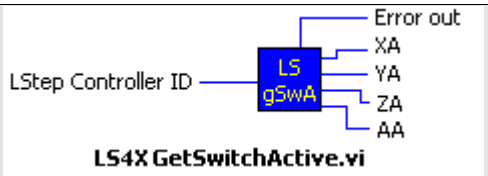
LS_SetAutoLimitAfterCalibRM	
Beschreibung:	Verhindert, dass beim Kalibrieren und Tischhubmessendie interne Software-Limits gesetzt werden.
Delphi:	function LS_SetAutoLimitAfterCalibRM(lFlags: Integer): Integer; function LSX_ SetAutoLimitAfterCalibRM(LSID: Integer; lFlags: Integer): Integer;
C++:	int SetAutoLimitAfterCalibRM (int lFlags);
LabView:	
Parameter:	lFlags: Bit-Maske Bit 0 = 1 → Bei der x-Achse werden keine Verfahrbereichsgrenzen gesetzt Bit 1 = 0 → Für die y-Achse werden Software-Limits gesetzt (calib/rm)
Beispiel:	LS. SetAutoLimitAfterCalibRM(lFlags);

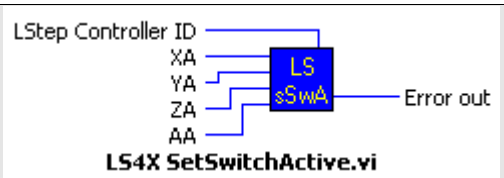
LS_GetLimit	
Beschreibung:	Verfahrbereichsgrenzen lesen
Delphi:	function LS_GetLimit(Axis: Integer; var MinRange, MaxRange: Double): Integer; function LSX_GetLimit(LSID: Integer; Axis: Integer; var MinRange, MaxRange: Double): Integer;
C++:	int GetLimit (int lAxis, double *pdMinRange, double *pdMaxRange);
LabView:	
Parameter:	Axis: die Achse, welcher Verfahrbereichsgrenzen gelesen werden sollen (X, Y, Z, A numeriert von 1 bis 4) MinRange: untere Verfahrbereichsgrenze, abhängig von der Dimension MaxRange: obere Verfahrbereichsgrenze, abhängig von der Dimension
Beispiel:	LS.GetLimit(1, &MinRange, &MaxRange);

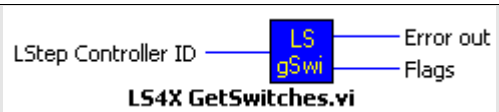
LS_SetLimit	
Beschreibung:	Verfahrbereichsgrenzen einstellen
Delphi:	function LS_SetLimit(Axis: Integer; MinRange, MaxRange: Double): Integer; function LSX_SetLimit(LSID: Integer; Axis: Integer; MinRange, MaxRange: Double): Integer;
C++:	int SetLimit (int lAxis,double dMinRange,double dMaxRange);
LabView:	
Parameter:	Axis: die Achse, welcher Verfahrbereichsgrenzen zugewiesen werden sollen (X, Y, Z, A numeriert von 1 bis 4) MinRange: untere Verfahrbereichsgrenze MaxRange: obere Verfahrbereichsgrenze
Beispiel:	LS.SetLimit(1, -10.0, 20.0); (Achse X –10 als untere und 20 als obere Verfahrbereichsgrenze zuweisen)

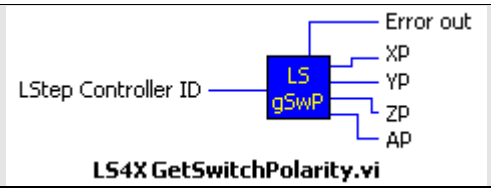
LS_GetLimitControl	
Beschreibung:	Liest, ob die Bereichsüberwachung aktiv ist
Delphi:	function LS_GetLimitControl(Axis: Integer; var Active: LongBool): Integer; function LSX_GetLimitControl(LSID: Integer; Axis: Integer; var Active: LongBool): Integer;
C++:	int GetLimitControl (int lAxis, BOOL *pbActive);
LabView:	
Parameter:	Active: True = Bereichsüberwachung der jeweiligen Achse ist aktiv
Beispiel:	LS.GetLimitControl(2, &Active); // Activ = False heißt: Bereichsüberwachung von Achse y ist deaktiviert.

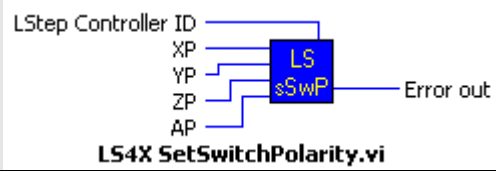
LS_SetLimitControl	
Beschreibung:	Bereichsüberwachung
Delphi:	function LS_SetLimitControl(Axis: Integer; Active: LongBool): Integer; function LSX_SetLimitControl(LSID: Integer; Axis: Integer; Active: LongBool): Integer;
C++:	int SetLimitControl (int lAxis,BOOL Active);
LabView:	
Parameter:	Axis: (X, Y, Z, A numeriert von 1 bis 4) Active: Bereichsüberwachung der jeweiligen Achse aktivieren
Beispiel:	LS.SetLimitControl(2, true); // Bereichsüberwachung von Achse y aktiv

LS_GetSwitchActive	
Beschreibung:	Liest Status für Endschalter Ein / Aus
Delphi:	function LS_GetSwitchActive(var XA, YA, ZA, AA: Integer): Integer; function LSX_GetSwitchActive(LSID: Integer; var XA, YA, ZA, AA: Integer): Integer;
C++:	int GetSwitchActive (int *plXA, int *plYA, int *plZA, int *plRA);
LabView:	
Parameter:	Für jede Achse wird eine Bitmaske geliefert:
	Bit 0 ➔ Null-Endschalter Bit 1 ➔ Referenz-Endschalter Bit 2 ➔ End-Endschalter
	Ist das Bit gesetzt - ist der jeweilige Schalter aktiv.
Beispiel:	LS.GetSwitchActive(&XA, &YA, &ZA, &RA);

LS_SetSwitchActive	
Beschreibung:	Endschalter Ein/Aus
Delphi:	function LS_SetSwitchActive(XA, YA, ZA, AA: Integer): Integer; function LSX_SetSwitchActive(LSID: Integer; XA, YA, ZA, AA: Integer): Integer;
C++:	int SetSwitchActive (int lXA,int lYA,int lZA,int lAA);
LabView:	
Parameter:	Für jede Achse wird eine Bitmaske übergeben:
	Bit 0 ➔ Null-Endschalter Bit 1 ➔ Referenz-Endschalter Bit 2 ➔ End-Endschalter
	Um den jeweiligen Schalter zu aktivieren, muß das Bit gesetzt werden.
Beispiel:	LS.SetSwitchActive(7, 1, 5, 0); (Alle Endschalter der X-Achse Ein; Null-Endschalter der Y-Achse Ein; E0 und EE der Z-Achse ein; A-Achse: Alle Endschalter Aus)

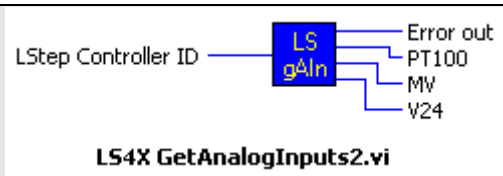
LS_GetSwitches	
Beschreibung:	liest den Zustand aller Endschalter
Delphi:	function LS_GetSwitches(var Flags: Integer): Integer;
C++:	int GetSwitches (int *plFlags);
LabView:	
Parameter:	Value: Zeiger auf Integerwert, der den Zustand aller Endschalter als Bitmaske enthält. In der Bitmaske ist der Zustand der Endschalter wie folgt verschlüsselt: Endschalter EE Ref. E0 Achse AZYX AZYX AZYX Bit 0000 0000 0000 z.B. Flags = 0x003 → E0 von X- und Y-Achse sind angefahren Flags = 0x200 → EE von Y-Achse ist angefahren
Beispiel:	LS.GetSwitches(&Flags);

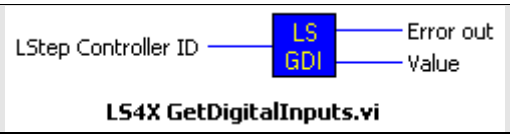
LS_GetSwitchPolarity	
Beschreibung:	Liest Endschalterpolarität
Delphi:	function LS_GetSwitchPolarity(var XP, YP, ZP, AP: Integer): Integer; function LSX_GetSwitchPolarity(LSID: Integer; var XP, YP, ZP, AP: Integer): Integer;
C++:	int GetSwitchPolarity (int *plXP, int *plYP, int *plZP, int *plRP);
LabView:	
Parameter:	Für jede Achse wird eine Bitmaske geliefert:
	Bit 0 ➔ Null-Endschalter Bit 1 ➔ Referenz-Endschalter Bit 2 ➔ End-Endschalter
	Ist das Bit gesetzt werden - reagiert der jeweilige Schalter auf die positive Flanke
Beispiel:	LS.GetSwitchPolarity(&XP, &YP, &ZP, &RP);

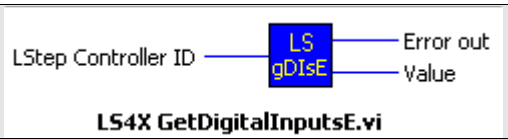
LS_SetSwitchPolarity	
Beschreibung:	Endschalterpolarität einstellen
Delphi:	function LS_SetSwitchPolarity(XP, YP, ZP, AP: Integer): Integer; function LSX_SetSwitchPolarity(LSID: Integer; XP, YP, ZP, AP: Integer): Integer;
C++:	int SetSwitchPolarity (int lXP,int lYP,int lZP,int lRA);
LabView:	
Parameter:	Für jede Achse wird eine Bitmaske übergeben:
	Bit 0 ➔ Null-Endschalter Bit 1 ➔ Referenz-Endschalter Bit 2 ➔ End-Endschalter
	Reagiert der jeweilige Schalter auf die positive Flanke, muß das Bit gesetzt werden.
Beispiel:	LS.SetSwitchPolarity(7, 0, 0, 0); (Alle Endschalter der X-Achse High-Aktiv, alle Endschalter der Y-Achse Low-Aktiv...)

Digitale und analoge Ein.- und Ausgänge

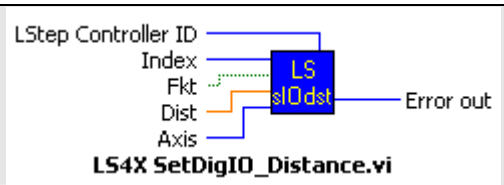
LS_GetAnalogInput	
Beschreibung:	Lesen des aktuellen Zustands eines Analogkanals
Delphi:	function LS_GetAnalogInput(Index: Integer; var Value: Integer): Integer; function LSX_GetAnalogInput(LSID: Integer; Index: Integer; var Value: Integer): Integer;
C++:	int GetAnalogInput (int lIndex,int *plValue);
LabView:	
Parameter:	Index: 0-9 (Analogkanäle) Value: Zeiger auf Integerwert, der den aktuellen Zustand des Analogkanals angibt
Beispiel:	LS.GetAnalogInput(0, &Eingang0);

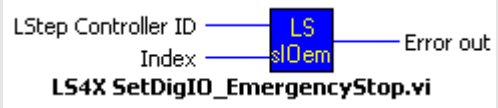
LS_GetAnalogInputs2	
Beschreibung	Lesen der aktuellen Zustände der Analogkanäle (Channel 6, 7, 8) nur bei der LSTEP-PCI
Delphi	function LS_GetAnalogInputs2(var PT100, MV, V24: Integer): Integer; function LSX_GetAnalogInputs2(LSID: Integer; var PT100, MV, V24: Integer): Integer;
C++	int GetAnalogInputs2 (int *plPT100, int *plMV, int *plV24);
LabView	
Parameter	PT100, MV, V24: Zeiger auf Integerwert, in den GetAnalogInputs2 den aktuellen Zustand des Analogkanals schreiben soll
Beispiel	LS.GetAnalogInputs2(&PT100, &MV, &V24);

LS_GetDigitalInputs	
Beschreibung:	Alle Inputpins lesen
Delphi:	function LS_GetDigitalInputs(var Value: Integer): Integer; function LSX_GetDigitalInputs(LSID: Integer; var Value: Integer): Integer;
C++:	int GetDigitalInputs (int *plValue);
LabView:	
Parameter:	Value: Zeiger auf Integerwert, der den Zustand aller Eingänge als Bitmaske enthält.
Beispiel:	int Eingange; LS.GetDigitalInputs(&Eingange); if (Eingange & 16) ... // Wenn Inputpin 4 gesetzt

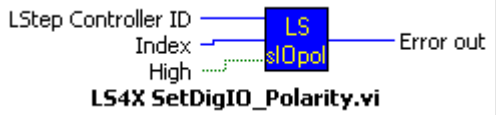
LS_GetDigitalInputsE	
Beschreibung	Zusätzliche digitale Eingänge lesen (16-31)
Delphi	function LS_GetDigitalInputsE(var Value: Integer): Integer; function LSX_GetDigitalInputsE(LSID: Integer; var Value: Integer): Integer;
C++	int GetDigitalInputsE (int *plValue);
LabView:	
Parameter	Value: Zeiger auf einen 32-bit-Integer, welcher nach Aufruf der Funktion in den Bits 0-15 den Status der Eingänge 16-31 enthält.
Beispiel	LS.GetDigitalInputsE(i);

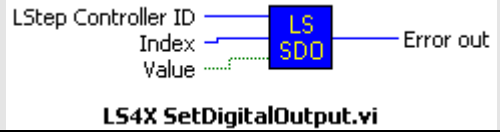
LS_SetAnalogOutput	
Beschreibung:	Analogkanal setzen
Delphi:	function LS_SetAnalogOutput(Index: Integer; Value: Integer): Integer; function LSX_SetAnalogOutput(LSID: Integer; Index: Integer; Value: Integer): Integer;
C++:	int SetAnalogOutput (int lIndex,int lValue);
LabView:	
Parameter:	Index: 0-1 (Analogkanäle) Value: 0-100 [%]
Beispiel:	LS.SetAnalogOutput(0, 100); // Ausgang 0 auf Maximum setzen

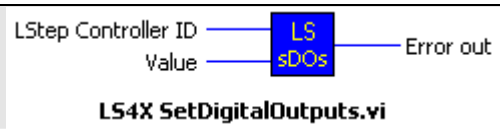
LS_SetDigIO_Distance	
Beschreibung:	Funktion der digitalen Ein-/Ausgänge
	Aktivierung eines Ausgang in Abhängigkeit der eingestellten Strecke vor/nach der Zielposition.
Delphi:	function LS_SetDigIO_Distance(Index: Integer; Fkt: LongBool; Dist: Double; Axis: Integer): Integer; function LSX_SetDigIO_Distance(LSID: Integer; Index: Integer; Fkt: LongBool; Dist: Double; Axis: Integer): Integer;
C++:	int SetDigIO_Distance (int lIndex,BOOL Fkt,double dDist,int lAxis);
LabView:	
Parameter:	Index: 0 bis 15 (Outputpin)
	Fkt = false ➔ Aktivierung eines Ausgang in Abhängigkeit der eingestellten Strecke vor der Zielposition. Fkt = true ➔ Aktivierung eines Ausgang in Abhängigkeit der eingestellten Strecke nach der Startposition.
	Dist: Strecke (Eingabe ist abhängig von der eingestellten Dimension) Axis: (X, Y, Z, A numeriert von 1 bis 4)
Beispiel:	LS.SetDigIO_Distance(7, false, 78.9, 3); /* Ausgang 7 wird 78.9mm vor erreichen der Zielposition (Z- Achse) aktiviert. */

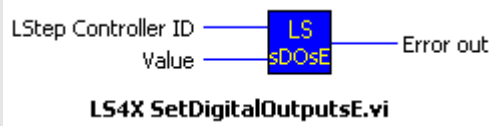
LS_SetDigIO_EmergencyStop	
Beschreibung:	Funktion der digitalen Ein-/Ausgänge Zuordnung des Not-Stop-Pins
Delphi:	function LS_SetDigIO_EmergencyStop(Index: Integer): Integer; function LSX_SetDigIO_EmergencyStop(LSID: Integer; Index: Integer): Integer;
C++:	int SetDigIO_EmergencyStop (int lIndex);
LabView:	
Parameter:	Index: 0 bis 15 (Input/Output)
Beispiel:	LS.SetDigIOEmergencyStop(15); // Not-Stop-Pin 15

LS_SetDigIO_Off	
Beschreibung:	Funktion der digitalen Ein-/Ausgänge Aus (Keine Beeinflussung der Ein-/Ausgänge)
Delphi:	function LS_SetDigIO_Off(Index: Integer): Integer; function LSX_SetDigIO_Off(LSID: Integer; Index: Integer): Integer;
C++:	int SetDigIO_Off (int lIndex);
LabView:	
Parameter:	Index: 0 bis 15 (Input/Output), 16 (alle 16 Portpins)
Beispiel:	LS.SetDigIO_Off(0); // dig. Fkt. Input-/Outputpin 0 Aus

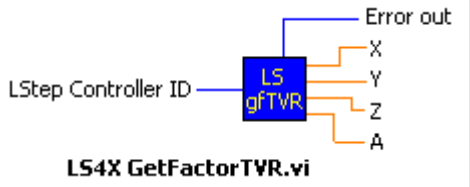
LS_SetDigIO_Polarity	
Beschreibung:	Funktion der digitalen Ein-/Ausgänge Einstellung der Polarität
Delphi:	function LS_SetDigIO_Polarity(Index: Integer; High: LongBool): Integer; function LSX_SetDigIO_Polarity(LSID: Integer; Index: Integer; High: LongBool): Integer;
C++:	int SetDigIO_Polarity (int lIndex,BOOL High);
LabView:	
Parameter:	Index: 0 bis 15 (Input/Output), 16 (alle 16 Portpins)
	High = true ➔ High-Aktiv High = false ➔ Low-Aktiv
Beispiel:	LS.SetDigIO_Polarity(3, True); // Input-/Outputpin 3 High-Aktiv

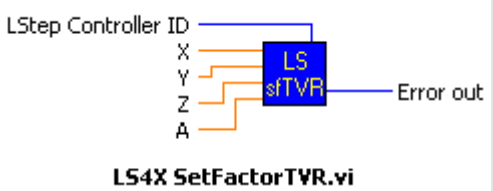
LS_SetDigitalOutput	
Beschreibung:	Outputpin setzen
Delphi:	function LS_SetDigitalOutput(Index: Integer; Value: LongBool): Integer; function LSX_SetDigitalOutput(LSID: Integer; Index: Integer; Value: LongBool): Integer;
C++:	int SetDigitalOutput (int lIndex,BOOL Value);
LabView:	
Parameter:	Index: 0-15 Value: Zustand auf „0“ oder „1“ setzen
Beispiel:	LS.SetDigitalOutput(0, true); // Outputpin 0 auf „1“ setzen

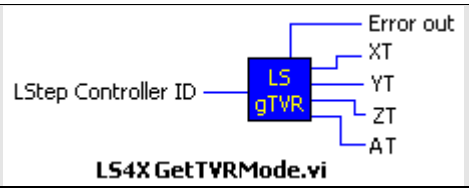
LS_SetDigitalOutputs	
Beschreibung	Digitale Ausgänge setzen (0-15)
Delphi	function LS_SetDigitalOutputs(Value: Integer): Integer; function LSX_SetDigitalOutputs(LSID: Integer; Value: Integer): Integer;
C++	int SetDigitalOutputs (int lValue);
LabView:	
Parameter	Value: Bitmaske, der Wert auf den die Ausgänge 0-15 gesetzt werden, wird durch die Bits 0-15 bestimmt.
Beispiel	LS.SetDigitalOutputs($03); // Ausgänge 0 und 1 auf 1 setzen, die übrigen auf 0

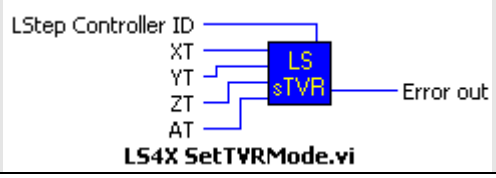
LS_SetDigitalOutputsE	
Beschreibung	Zusätzliche digitale Ausgänge setzen (16-31)
Delphi	function LS_SetDigitalOutputsE(Value: Integer): Integer; function LSX_SetDigitalOutputsE(LSID: Integer; Value: Integer): Integer;
C++	int SetDigitalOutputsE (int lValue);
LabView:	
Parameter	Value: Bitmaske, der Wert auf den die Ausgänge 16-31 gesetzt werden, wird durch die Bits 0-15 bestimmt.
Beispiel	LS.SetDigitalOutputsE($03); // Ausgänge 16 und 17 auf 1 setzen, die übrigen auf 0

Takt-Vor/Rück Eingänge

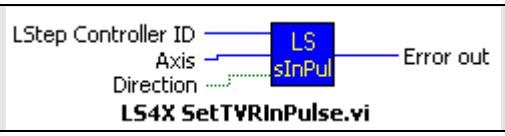
LS_GetFactorTVR	
Beschreibung:	Liest den Faktor Takt Vor / Rück
Delphi:	function LS_GetFactorTVR(var X, Y, Z, A: Double): Integer; function LSX_GetFactorTVR(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetFactorTVR (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z und A: Factor Takt Vor/Rück. Z. B. X = 10 heißt: Ein Takt = zehn Motorinkremente
Beispiel:	LS.GetFactorTVR(&X, &Y, &Z, &A);

LS_SetFactorTVR	
Beschreibung:	Faktor Takt Vor / Rück
Delphi:	function LS_SetFactorTVR(X, Y, Z, A: Double): Integer; function LSX_SetFactorTVR(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetFactorTVR (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z und A 0.01 – 100.00
Beispiel:	LS.SetFactorTVR(2.0, 2.0, 0, 0); /* Bei Achse X und Y soll Takt Vor/Rück mit dem Faktor 2 arbeiten */

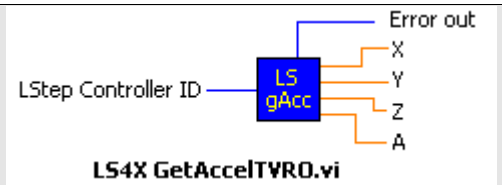
LS_GetTVRMode	
Beschreibung:	Einstellung vom Takt Vor / Rück lesen
Delphi:	function LS_ GetTVRMode(var XT, YT, ZT, AT: Integer): Integer; function LSX_ GetTVRMode(LSID: Integer; var XT, YT, ZT, AT: Integer): Integer;
C++:	int GetTVRMode (int *plXT, int *plYT, int *plZT, int *plRT);
LabView:	
Parameter:	TVR-Modus von X, Y, Z und A: 0 ➔ Takt Vor/Rück ist „AUS“ 1 ➔ Normale Takt Vor/Rück Bearbeitung 2 ➔ Takt Vor/Rück Bearbeitung arbeitet mit einem Faktor 3 ➔ Takt Vor/Rück Bearbeitung benötigt externe Freigabe über Triggerout Pin (MFP). 4 ➔ Kombination aus 2 & 3.
Beispiel:	LS. GetTVRMode(&XT, &YT, &ZT, &RT);

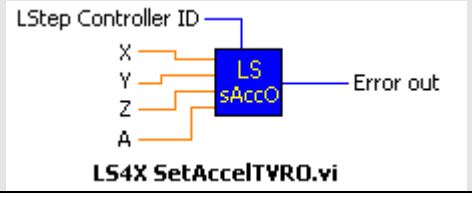
LS_SetTVRMode	
Beschreibung:	Takt Vor / Rück einstellen
Delphi:	function LS_SetTVRMode(XT, YT, ZT, AT: Integer): Integer; function LSX_SetTVRMode(LSID: Integer; XT, YT, ZT, AT: Integer): Integer;
C++:	int SetTVRMode (int lXT,int lYT,int lZT,int lAT);
LabView:	
Parameter:	TVR-Modus von X, Y, Z und A: 0 ➔ Takt Vor/Rück „AUS“ 1 ➔ Normale Takt Vor/Rück Bearbeitung 2 ➔ Takt Vor/Rück Bearbeitung arbeitet mit einem Faktor 3 ➔ Takt Vor/Rück Bearbeitung benötigt externe Freigabe über Triggerout Pin (MFP). 4 ➔ Kombination aus 2 & 3.
Beispiel:	LS.SetTVRMode(1, 1, 0, 0); // TVR X- und Y-Achse Ein

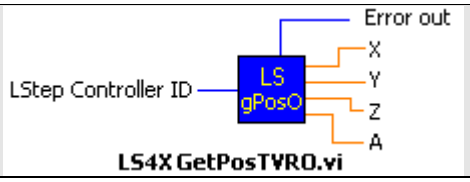
Takt-Vor/Rück über Schnittstelle

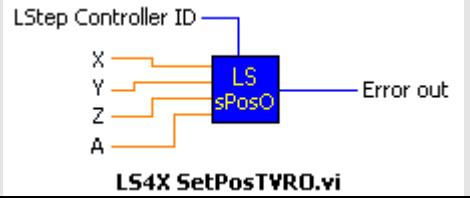
LS_SetTVRInPulse	
Beschreibung:	Diese Funktion hat die gleiche Auswirkung wie ein externer Takt mit Richtungsinformation.
Delphi:	function LS_SetTVRInPulse (Axis: Integer; Direction: Boolean): Integer; function LSX_SetTVRInPulse (LSID: Integer; Axis: Integer; Direction: Boolean): Integer;
C++:	int SetTVRInPulse (int Axis, BOOL Direction);
LabView:	
Parameter:	Wert: Anzahl der ausgeführten Trigger
Beispiel:	LS.SetTVRInPulse (2, true); // 1 Takt vorwärts bei der y-Achse.

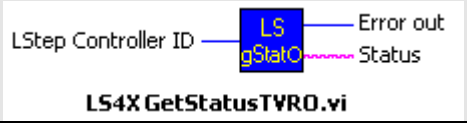
Takt-Vor/Rück Ausgänge für weitere Achsen

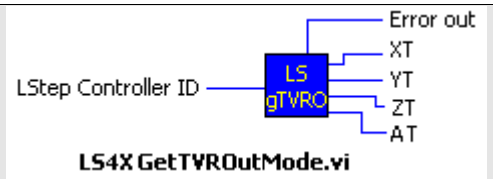
LS_GetAccelTVRO	
Beschreibung:	Liest die eingestellten Beschleunigungen für die weiteren Achsen
Delphi:	function LS_GetAccelTVRO(var X, Y, Z, A: Double): Integer; function LSX_GetAccelTVRO(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetAccelTVRO (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Beschleunigungswerte, U/s²
Beispiel:	LS.GetAccelTVRO(&X, &Y, &Z, &A);

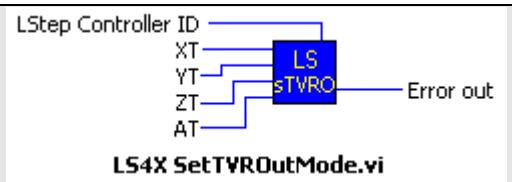
LS_SetAccelTVRO	
Beschreibung:	Beschleunigung für die weiteren Achsen einstellen
Delphi:	function LS_SetAccelTVRO(X, Y, Z, A: Double): Integer; function LSX_SetAccelTVRO(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetAccelTVRO (double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z und A: Beschleunigungen, Wertebereich 0.01 – 1500 [U/s²]
Beispiel:	LS.SetAccelTVRO(1.0, 1.5, 0, 0);

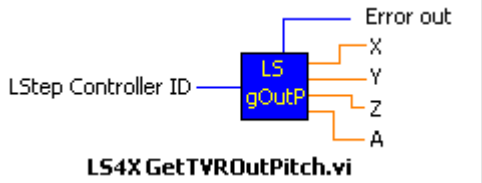
LS_GetPosTVRO	
Beschreibung:	Position der zusätzlichen Achsen ablesen
Delphi:	function LS_GetPosTVRO(var dX, dY, dZ, dR: Double): Integer; function LSX_GetPosTVRO(LSID: Integer; var dX, dY, dZ, dR: Double): Integer;
C++:	int GetPosTVRO (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	dX, dY, dZ, dR: Positionswert, abhängig von der Dimension
Beispiel:	LS. GetPosTVRO(&X, &Y, &Z, &R);

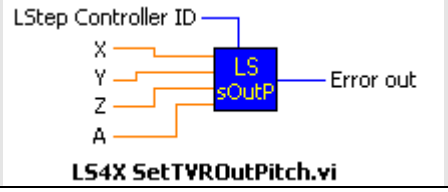
LS_SetPosTVRO	
Beschreibung:	Position der zusätzlichen Achsen setzen
Delphi:	function LS_ SetPosTVRO(dX, dY, dZ, dR: Double): Integer; function LSX_ SetPosTVRO(LSID: Integer; dX, dY, dZ, dR: Double): Integer;
C++:	int SetPosTVRO (double dX, double dY, double dZ, double dR);
LabView:	
Parameter:	dX, dY, dZ, dR: Positionswert, in Abhängigkeit von Dimension Wertebereich: min. Bereichsgrenze bis max. Bereichsgrenze
Beispiel:	LS. SetPosTVRO(10.0, 5.0, 0.0, 0.0);

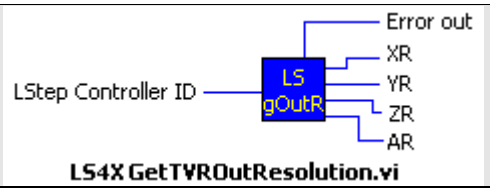
LS_GetStatusTVRO	
Beschreibung:	Liefert den aktuellen Status der zusätzlichen Achsen
Delphi:	function LS_GetStatusTVRO(pcStat: PChar; MaxLen: Integer): Integer; function LSX_ GetStatusTVRO(LSID: Integer; pcStat: PChar; MaxLen: Integer): Integer;
C++:	int GetStatusTVRO (char *pcStat, int lMaxLen);
LabView:	
Parameter:	pcStat: Zeiger auf einen Puffer, in dem der Statusstring zurückgegeben wird MaxLen: Maximale Anzahl von Zeichen, die in den Puffer kopiert werden dürfen z.B.: @ - M – @ = Achse steht M = Achse ist in Bewegung (Motion) - = Achse ist nicht freigegeben
Beispiel:	LS. GetStatusTVRO(pcStat, 256); // Die zusätzliche Z-Achse um 5mm in positiver Richtung verfahren

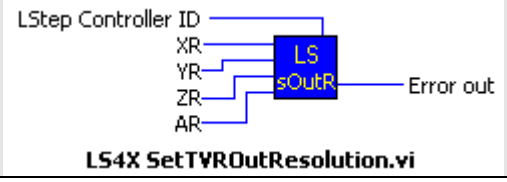
LS_GetTVROutMode	
Delphi:	function LS_GetTVROutMode(var X, Y, Z, A: Integer): Integer; function LSX_GetTVROutMode(LSID: Integer; var X, Y, Z, A: Integer): Integer;
C++:	int GetTVROutMode (int *plXT, int *plYT, int *plZT, int *plAT);
LabView:	
Parameter:	X, Y, Z und A: 0 => Takt Vor/Rück ist "AUS" 1 => Takt Vor/Rück ist "EIN"
Beispiel:	LS.GetTVROutMode(&X, &Y, &Z, &A);

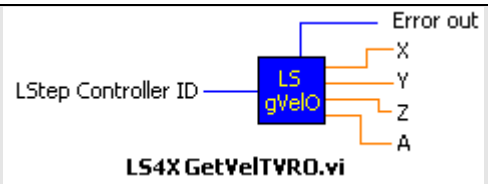
LS_SetTVROutMode	
Beschreibung:	Zusetzliche Achsen X, Y, Z und A, neben den eigentlichen Hauptachsen X, Y, Z und A, setzen
Delphi:	function LS_SetTVROutMode(X, Y, Z, A: Integer): Integer; function LSX_SetTVROutMode(LSID: Integer; X, Y, Z, A: Integer): Integer;
C++:	int SetTVROutMode (int lXT, int lYT, int lZT, int lAT);
LabView:	
Parameter:	X, Y, Z und A: 0 oder 1
Beispiel:	LS.SetTVROutMode(1, 0, 1, 0); //Bei Achsen x und z soll Takt Vor/Rück aktiviert werden, bei y und a – deaktiviert.

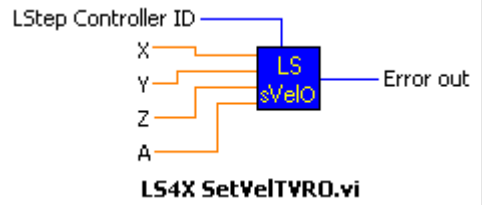
LS_GetTVROutPitch	
Beschreibung:	Liest die Spindelsteigungen für die zusätzlichen Achsen
Delphi:	function LS_GetTVROutPitch(var X, Y, Z, R: Double): Integer; function LSX_ GetTVROutPitch (LSID: Integer; var X, Y, Z, R: Double): Integer;
C++:	int GetTVROutPitch (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z und R: Spindelsteigungen [mm]
Beispiel:	LS. GetTVROutPitch(&X, &Y, &Z, &A);

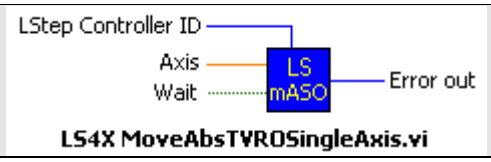
LS_SetTVROutPitch	
Beschreibung:	Setzt die Spindelsteigungen für die zusätzlichen Achsen
Delphi:	function LS_SetTVROutPitch(X, Y, Z, R: Double): Integer; function LSX_ SetTVROutPitch (LSID: Integer; X, Y, Z, R: Double): Integer;
C++:	int SetTVROutPitch (double dX, double dY, double dZ, double dR);
LabView:	
Parameter:	X, Y, Z und R: Spindelsteigungen [mm], Wertebereich 0.001 bis 100
Beispiel:	LS. SetTVROutPitch(1.0, 4.0, 1.0, 1.0); /* Spindelsteigung für y-Achse beträgt 4 mm. Für x-, z- und a-Achsen werden Spindeln mit 1mm Steigung eingesetzt*/

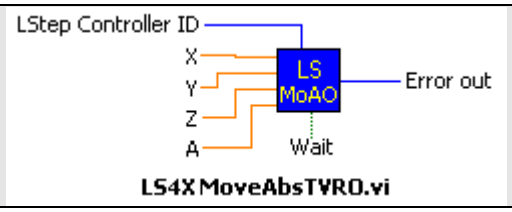
LS_GetTVROutResolution	
Beschreibung:	Liest die Auflösung der an zu steuernden Endstufe
Delphi:	function LS_GetTVROutResolution(var X, Y, Z, A: Integer): Integer; function LSX_ GetTVROutResolution (LSID: Integer; var X, Y, Z, A: Integer): Integer;
C++:	int GetTVROutResolution (int *plX, int *plY, int *plZ, int *plA);
LabView:	
Parameter:	X, Y, Z und A: Impulsen pro Umdrehung
Beispiel:	LS. GetTVROutResolution (&X, &Y, &Z, &A);

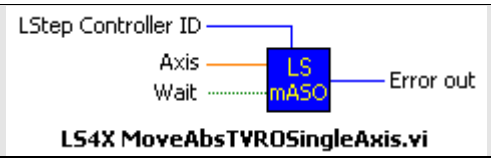
LS_SetTVROutResolution	
Beschreibung:	Auflösung der an zu steuernden Endstufe einstellen
Delphi:	function LS_SetTVROutResolution(X, Y, Z, A: Integer): Integer; function LSX_ SetTVROutResolution (LSID: Integer; X, Y, Z, A: Integer): Integer;
C++:	int SetTVROutResolution (int lX, int lY, int lZ, int lA);
LabView:	
Parameter:	X, Y, Z und A: Impulsen pro Umdrehung, Wertebereich 0 bis 51200
Beispiel:	LS. SetTVROutResolution (1000, 1000, 0, 0); /* Bei Achse X und Y wird eine Auflösung von 1000 Impulsen pro Umdrehung eingestellt*/

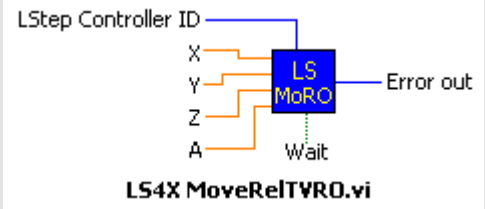
LS_GetVelTVRO	
Beschreibung:	Liest die eingestellten Geschwindigkeiten für die weiteren Achsen
Delphi:	function LS_GetVelTVRO(var X, Y, Z, A: Double): Integer; function LSX_GetVelTVRO(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetVelTVRO (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z, A: Geschwindigkeitswerte, [U/s]
Beispiel:	LS.GetVelTVRO(&X, &Y, &Z, &A);

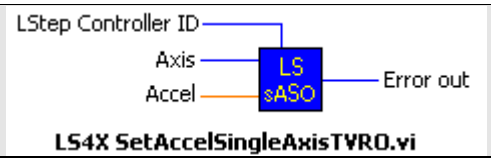
LS_SetVelTVRO	
Beschreibung:	Geschwindigkeit für die weiteren Achsen einstellen
Delphi:	function LS_SetVelTVRO(X, Y, Z, A: Double): Integer; function LSX_SetVelTVRO(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetVelTVRO (double dX, double dY, double dZ, double dA);
LabView:	
Parameter:	X, Y, Z und A: Geschwindigkeiten, 0 – 40.0 [U/s]
Beispiel:	LS.SetVelTVRO(1.0, 1.5, 0, 0);

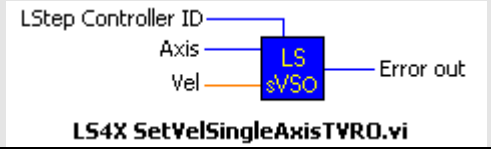
LS_MoveAbsTVROSingleAxis	
Beschreibung:	Einzelne Achse absolut positionieren
Delphi:	function LS_MoveAbsTVROSingleAxis (Axis: Integer; Value: Double; Wait: LongBool): Integer; function LSX_MoveAbsTVROSingleAxis (LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;
C++:	int MoveAbsTVROSingleAxis (int lAxis, double dValue, BOOL bWait);
LabView:	
Parameter:	Axis: (X, Y, Z, A numeriert von 1 bis 4) Value: Position (Eingabe ist abhängig von der eingestellten Dimension)
Beispiel:	LS.MoveAbsTVROSingleAxis (2, 10.0); //Die zusätzliche Y-Achse auf 10mm absolut positionieren

LS_MoveAbsTVRO	
Beschreibung:	Absolutposition anfahren
	(Die zusätzlichen Achsen x, y, z und a werden auf die übergebenen Positionswerte positioniert.)
Delphi:	function LS_MoveAbsTVRO(X, Y, Z, A: Double; Wait: LongBool): Integer; function LSX_MoveAbsTVRO (LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;
C++:	int MoveAbsTVRO (double dX, double dY, double dZ, double dR, BOOL bWait);
LabView:	
Parameter:	X, Y, Z und A +- Verfahrbereich Eingabe ist abhängig von der Dimension Wait: Gibt an, ob die Funktion nachdem die Position erreicht wurde (= true) oder direkt zurückkehren soll (= false)
Beispiel:	LS.MoveAbsTVRO (10.0, 10.0, 10.0, 10.0, true);

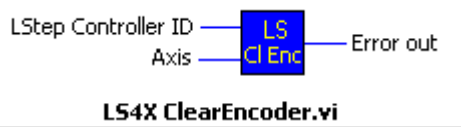
LS_MoveRelTVROSingleAxis	
Beschreibung:	Einzelne Achse relativ verfahren
Delphi:	function LS_MoveRelTVROSingleAxis (Axis: Integer; Value: Double; Wait: LongBool): Integer; function LSX_MoveRelTVROSingleAxis (LSID: Integer; Axis: Integer; Value: Double; Wait: LongBool): Integer;
C++:	int MoveRelTVROSingleAxis (int lAxis,double dValue,BOOL Wait);
LabView:	
Parameter:	Axis: (X, Y, Z, A numeriert von 1 bis 4) Value: Strecke (Eingabe ist abhängig von der eingestellten Dimension)
Beispiel:	LS.MoveRelTVROSingleAxis (3, 5.0); // Die zusätzliche Z-Achse um 5mm in positiver Richtung verfahren

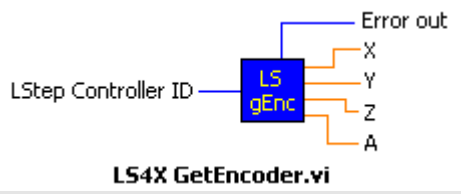
LS_MoveRelTVRO	
Beschreibung:	Relativen Vektor fahren (Die zusätzlichen Achsen x, y, z und a werden um die übergebenen Strecken verfahren.)
Delphi:	function LS_MoveRelTVRO(X, Y, Z, A: Double; Wait: LongBool): Integer; function LSX_MoveRelTVRO(LSID: Integer; X, Y, Z, A: Double; Wait: LongBool): Integer;
C++:	int MoveRelTVRO (double dX, double dY, double dZ, double dR, BOOL bWait);
LabView:	
Parameter:	X, Y, Z und A +- Verfahrbereich Eingabe ist abhängig von der Dimension Wait: Gibt an, ob die Funktion nachdem die Position erreicht wurde (= true) oder direkt zurückkehren soll (= false)
Beispiel:	LS.MoveRelTVRO(10.0, 10.0, 10.0, 10.0, true);

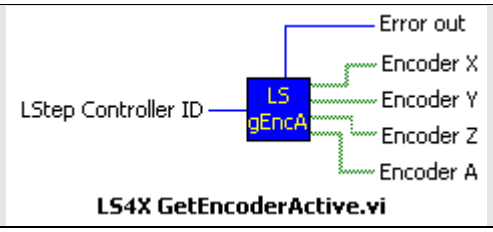
LS_SetAccelSingleAxisTVRO	
Beschreibung:	Beschleunigung für einzelne zusätzliche Achse einstellen
Delphi:	function LS_SetAccelSingleAxisTVRO(Axis: Integer; Accel: Double): Integer; function LSX_SetAccelSingleAxisTVRO (LSID: Integer; Axis: Integer; Accel: Double): Integer;
C++:	int SetAccelSingleAxisTVRO (int lAxis, double dAccel);
LabView:	
Parameter:	Axis: (X, Y, Z, A numeriert von 1 bis 4) Accel: 0.01 – 1500 [U/s²]
Beispiel:	LS.SetAccelSingleAxis(2, 50.0); // Die Z-Achse soll wird mit 50 U/s² beschleunigt

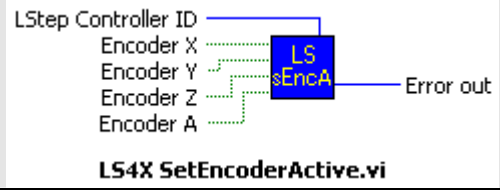
LS_SetVelSingleAxisTVRO	
Beschreibung:	Geschwindigkeit für einzelne zusätzliche Achse einstellen
Delphi:	function LS_SetVelSingleAxisTVRO(Axis: Integer; Vel: Double): Integer; function LSX_SetVelSingleAxisTVRO (LSID: Integer; Axis: Integer; Vel: Double): Integer;
C++:	int SetVelSingleAxisTVRO (int lAxis, double dVel);
LabView:	
Parameter:	Axis: (X, Y, Z, A numeriert von 1 bis 4) Vel: 0 – 40.0 [U/s]
Beispiel:	LS.SetVelSingleAxis(1, 10.0); // Die X-Achse soll mit einer max. Geschwindigkeit von 10 U/s betrieben werden

Geber-Einstellungen

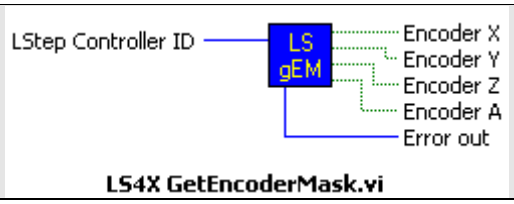
LS_ClearEncoder	
Beschreibung:	Geber-Zähler auf null setzen
Delphi:	function LS_ClearEncoder(lAxis: Integer): Integer; function LSX_ClearEncoder (LSID: Integer; lAxis: Integer): Integer;
C++:	int ClearEncoder (int lAxis);
LabView:	
Parameter:	lAxis: (X, Y, Z, A numeriert von 1 bis 4)
Beispiel:	LS. ClearEncoder (2); //Geber-Zähler der y-Achse auf null setzen

LS_GetEncoder	
Beschreibung:	Liest alle Geber-Positionen
Delphi:	function LS_GetEncoder(XP, YP, ZP, AP: Double): Integer; function LSX_GetEncoder (LSID: Integer; XP, YP, ZP, AP: Double): Integer;
C++:	int GetEncoder (double *pdXP, double *pdYP, double *pdZP, double *pdRP);
LabView:	
Parameter:	XP, YP, ZP, AP: Zählerwerte, 4-fach interpoliert
Beispiel:	LS. GetEncoder (&XP, &YP, &ZP, &AP);

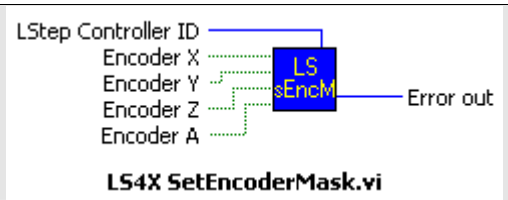
LS_GetEncoderActive	
Beschreibung:	Liest, welche Geber nach der Kalibration aktiviert werden.
Delphi:	function LS_GetEncoderActive(var Flags: Integer): Integer; function LSX_GetEncoderActive(LSID: Integer; var Flags: Integer): Integer;
C++:	int GetEncoderActive (int *plFlags);
LabView:	
Parameter:	Flags: Gebermaske
Beispiel:	LS.GetEncoderActive(&Flags);

LS_SetEncoderActive	
Beschreibung:	Mit dieser Funktion kann ausgewählt werden, welche Geber nach der Kalibration aktiviert werden sollen.
Delphi:	function LS_SetEncoderActive(Flags: Integer): Integer; function LSX_SetEncoderActive(LSID: Integer; Flags: Integer): Integer;
C++:	int SetEncoderActive (int lFlags);
LabView:	
Parameter:	Value: Gebermaske

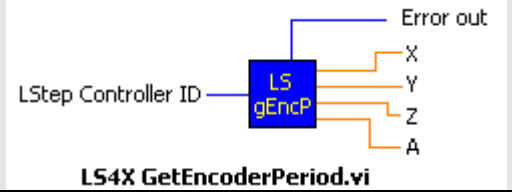
Beispiel:	LS.SetEncoderActive(0);
	// Alle Geber deaktivieren
	LS.SetEncoderMask(2);
	// Geber Y-Achse aktivieren

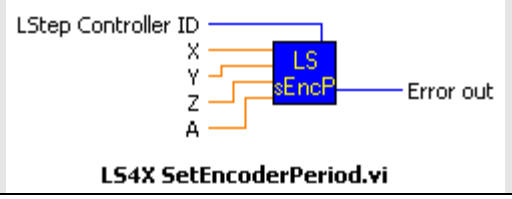
LS_GetEncoderMask	
Beschreibung:	Geberzustände auslesen
Delphi:	function LS_GetEncoderMask (var Flags: Integer): Integer; function LSX_GetEncoderMask(LSID: Integer; var Flags: Integer): Integer;
C++:	int GetEncoderMask (int *plFlags);
LabView:	
Parameter:	Flags: Gebermaske

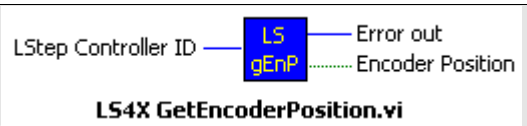
Beispiel:	int EncMask;
	LS.GetEncoderMask(&EncMask);
	if (EncMask & 2) ...
	// Wenn Geber Y-Achse angeschlossen+aktiv

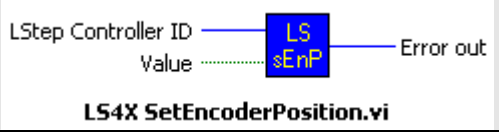
LS_SetEncoderMask	
Beschreibung:	Geber (de-)aktivieren
Delphi:	function LS_SetEncoderMask(Value: Integer): Integer; function LSX_SetEncoderMask(LSID: Integer; Value: Integer): Integer;
C++:	int SetEncoderMask (int lValue);
LabView:	
Parameter:	Value: Gebermaske

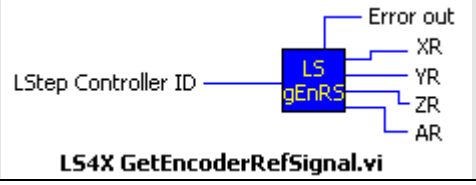
Beispiel:	LS.SetEncoderMask(0);
	// Alle Geber deaktivieren
	LS.SetEncoderMask(2);
	// Geber Y-Achse aktivieren

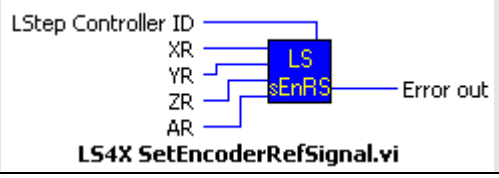
LS_GetEncoderPeriod	
Beschreibung:	Geberperiodenlängen auslesen
Delphi:	function LS_GetEncoderPeriod(var X, Y, Z, A: Double): Integer; function LSX_GetEncoderPeriod(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetEncoderPeriod (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z und A: Periodenlängen [mm]
Beispiel:	LS.GetEncoderPeriod(&X, &Y, &Z, &A);

LS_SetEncoderPeriod	
Beschreibung:	Geberperiodenlängen einstellen
Delphi:	function LS_SetEncoderPeriod(X, Y, Z, A: Double): Integer; function LSX_SetEncoderPeriod(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetEncoderPeriod (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z und A 0.0001 – Spindelsteigung * 0.8 (mm)
Beispiel:	LS.SetEncoderPeriod(0.1, 0.1, 0.1, 0.1); // Geberperiodenlänge aller Achsen ist 0.1mm

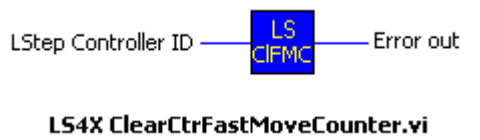
LS_GetEncoderPosition	
Beschreibung	Einstellung von der Geberwertanzeige lesen
Delphi	function LS_GetEncoderPosition(Value: Boolean): Integer; function LSX_GetEncoderPosition(LSID: Integer; Value: LongBool): Integer;
C++	int GetEncoderPosition (BOOL *pbValue);
LabView	
Parameter	Value: true → Bei der Positionsabfrage werden die Geberwerte der erkannten Geber angezeigt false → Geberpositionsanzeige ist "AUS"
Beispiel	LS.GetEncoderPosition(&Value);

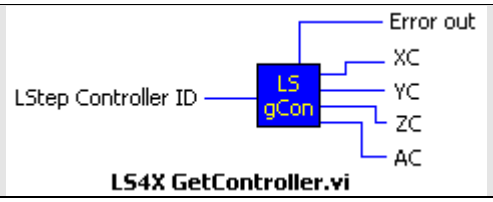
LS_SetEncoderPosition	
Beschreibung	Geberwertanzeige Ein/Aus
Delphi	function LS_SetEncoderPosition(Value: Boolean): Integer; function LSX_SetEncoderPosition(LSID: Integer; Value: LongBool): Integer;
C++	int SetEncoderPosition (BOOL fValue);
LabView	
Parameter	Value = true → Bei der Positionsabfrage werden die Geberwerte der erkannten Geber angezeigt
Beispiel	LS.SetEncoderPosition(true);

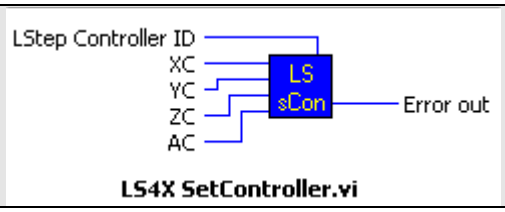
LS_GetEncoderRefSignal	
Beschreibung:	Liest, ob beim Kalibrieren Referenzsignal von Geber ausgewertet wird
Delphi:	function LS_GetEncoderRefSignal(var XR, YR, ZR, AR: Integer): Integer; function LSX_GetEncoderRefSignal(LSID: Integer; var XR, YR, ZR, AR: Integer): Integer;
C++:	int GetEncoderRefSignal (int *plXR, int *plYR, int *plZR, int *plRR);
LabView:	
Parameter:	X, Y, Z und A: 1 => Beim Kalibrieren wird das Referenzsignal ausgewertet 0 => Das Referenzsignal wird nicht ausgewertet
Beispiel:	LS.GetEncoderRefSignal(&X, &Y, &Z, &A);

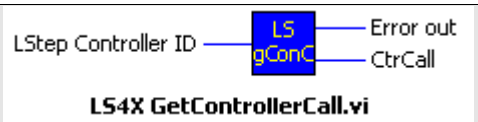
LS_SetEncoderRefSignal	
Beschreibung:	Beim Kalibrieren Referenzsignal von Geber auswerten
Delphi:	function LS_SetEncoderRefSignal(XR, YR, ZR, AR: Integer): Integer; function LSX_SetEncoderRefSignal(LSID: Integer; XR, YR, ZR, AR: Integer): Integer;
C++:	int SetEncoderRefSignal (int lXR,int lYR,int lZR,int lAR);
LabView:	
Parameter:	X, Y, Z und A 0 oder 1
Beispiel:	LS.SetEncoderRefSignal(1, 1, 0, 0); /* Beim Kalibrieren wird das Referenzsignal der Geber x und y ausgewertet. */

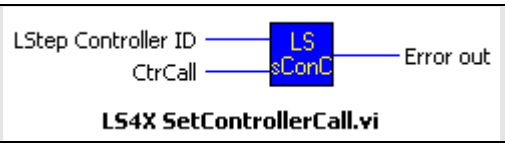
Reglereinstellungen

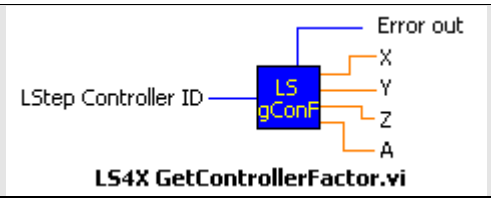
LS_ClearCtrFastMoveCounter	
Beschreibung:	Bei einer Reglerdifferenz, die größer als der Fangbereich ist, wird ein neuer Vektor gestartet und der dazu gehörige Counter um eins erhöht. Die Function setzt Fast Move Counters aller Achsen auf null.
Delphi:	function LS_ClearCtrFastMoveCounter: Integer; function LSX_ClearCtrFastMoveCounter(LSID: Integer): Integer;
C++:	int ClearCtrFastMoveCounter;
LabView:	
Parameter:	
Beispiel:	LS. ClearCtrFastMoveCounter;

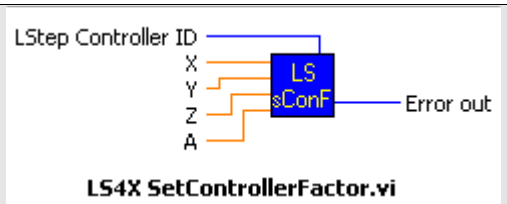
LS_GetController	
Beschreibung:	Regler-Modus auslesen
Delphi:	function LS_GetController(var XC, YC, ZC, AC: Integer): Integer; function LSX_GetController(LSID: Integer; var XC, YC, ZC, AC: Integer): Integer;
C++:	int GetController (int *plXC, int *plYC, int *plZC, int *plRC);
LabView:	
Parameter:	Regler-Modus X, Y, Z und A :
	0 ➔ Regler „AUS“
	1 ➔ Regler „AUS nach erreichen der Zielposition“
	2 ➔ Regler „Immer EIN“
	3 ➔ Regler „AUS nach erreichen der Zielposition“ mit reduziertem Strom
	4 ➔ Regler „Immer EIN“ mit reduziertem Strom
Beispiel:	LS.GetController(&X, &Y, &Z, &A);

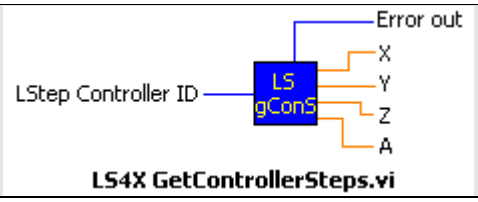
LS_SetController	
Beschreibung:	Regler-Modus einstellen
Delphi:	function LS_SetController(XC, YC, ZC, AC: Integer): Integer; function LSX_SetController(LSID: Integer; XC, YC, ZC, AC: Integer): Integer;
C++:	int SetController (int lXC,int lYC,int lZC,int lAC);
LabView:	
Parameter:	Regler-Modus X, Y, Z und A :
	0 ➔ Regler „AUS“
	1 ➔ Regler „AUS nach erreichen der Zielposition“
	2 ➔ Regler „Immer EIN“
zeit	3 ➔ Regler „AUS nach erreichen der Zielposition“ mit reduziertem Strom
	4 ➔ Regler „Immer EIN“ mit reduziertem Strom
Beispiel:	LS.SetController(1, 2, 0, 0);

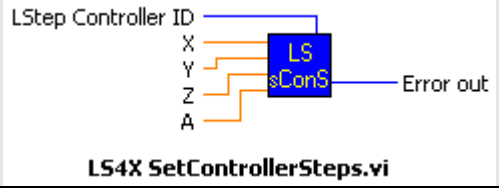
LS_GetControllerCall	
Beschreibung:	Liefert Regleraufrufzeit
Delphi:	function LS_GetControllerCall(var CtrCall: Integer): Integer; function LSX_GetControllerCall(LSID: Integer; var CtrCall: Integer): Integer;
C++:	int GetControllerCall (int *plCtrCall);
LabView:	
Parameter:	CtrCall: Regleraufrufzeit [ms]
Beispiel:	LS.GetControllerCall(&CtrCall); //Nach dem Funktionsaufruf CtrCall = 10 bedeutet: Regleraufruf alle 10 ms

LS_SetControllerCall	
Beschreibung:	Regleraufruf
Delphi:	function LS_SetControllerCall(CtrCall: Integer): Integer; function LSX_SetControllerCall(LSID: Integer; CtrCall: Integer): Integer;
C++:	int SetControllerCall (int lCtrCall);
LabView:	
Parameter:	CtrCall: Regleraufrufzeit [ms]
Beispiel:	LS.SetControllerCall(10);

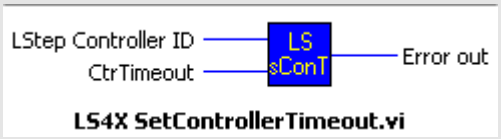
LS_GetControllerFactor	
Beschreibung:	Regler-Faktoren auslesen siehe Kap. 4.13 „Reglereinstellunen für LSTEP“
Delphi:	function LS_GetControllerFactor(var X, Y, Z, A: Double): Integer; function LSX_GetControllerFactor(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetControllerFactor (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z und A: Regler-Faktoren
Beispiel:	LS.GetControllerFactor(&X, &Y, &Z, &A);

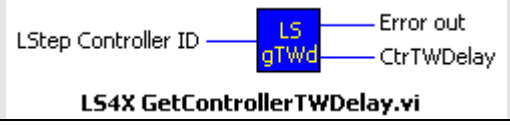
LS_SetControllerFactor	
Beschreibung:	Regler-Faktor
Delphi:	function LS_SetControllerFactor(X, Y, Z, A: Double): Integer; function LSX_SetControllerFactor(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetControllerFactor (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z und A 1 – 64
Beispiel:	LS.SetControllerFactor(1, 2, 3, 4);

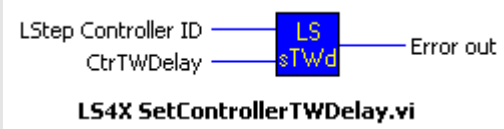
LS_GetControllerSteps	
Beschreibung:	Liefert die Regler-Schrittenlänge
Delphi:	function LS_GetControllerSteps(var X, Y, Z, A: Double): Integer; function LSX_GetControllerSteps(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetControllerSteps (double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter:	X, Y, Z und A: Regler-Schrittenlänge [mm]
Beispiel:	LS.GetControllerSteps(&X, &Y, &Z, &A);

LS_SetControllerSteps	
Beschreibung:	Regler-Schritte
Delphi:	function LS_SetControllerSteps(X, Y, Z, A: Double): Integer; function LSX_SetControllerSteps(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetControllerSteps (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z und A 1 – Spindelsteigung (Werte sind abhängig von der Dimension)
Beispiel:	LS.SetControllerSteps(4, 5, 7, 9);

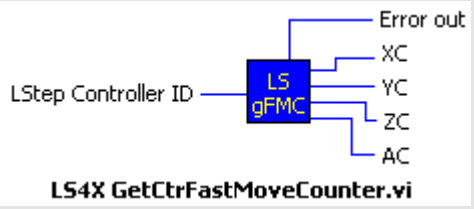
LS_GetControllerTimeout	
Beschreibung:	Liest Regler-Timeout
Delphi:	function LS_GetControllerTimeout(var ACtrTimeout: Integer): Integer; function LSX_GetControllerTimeout(LSID: Integer; var ACtrTimeout: Integer): Integer;
C++:	int GetControllerTimeout (int *plACtrTimeout);
LabView:	
Parameter:	ACtrTimeout: Timeout [ms], nach dem ein Verfahrbefehl mit Fehlermeldung (Fehlercode 4013) zurückkehrt, wenn die Steuerung eine Position nicht endgültig finden konnte.
Beispiel:	LS.GetControllerTimeout(&ACtrTimeout);

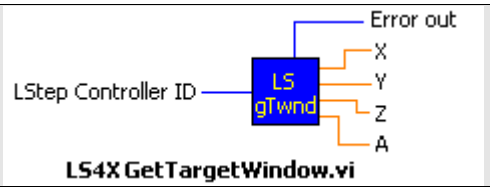
LS_SetControllerTimeout	
Beschreibung:	Regler-Timeout
Delphi:	function LS_SetControllerTimeout(ACtrTimeout: Integer): Integer; function LSX_SetControllerTimeout(LSID: Integer; ACtrTimeout: Integer): Integer;
C++:	int SetControllerTimeout (int ACtrTimeout);
LabView:	
Parameter:	ACtrTimeout: Timeout [ms], nach dem ein Verfahrbefehl mit Fehlermeldung (Fehlercode 4013) zurückkehrt, wenn die Steuerung eine Position nicht endgültig finden konnte.
Beispiel:	LS.SetControllerTimeout(500);

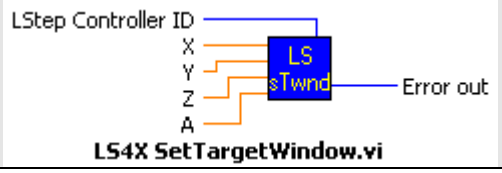
LS_GetControllerTWDelay	
Beschreibung	Regler-Verzögerung auslesen
Delphi	function LS_GetControllerTWDelay(var CtrTWDelay: Integer): Integer; function LSX_GetControllerTWDelay(LSID: Integer; var CtrTWDelay: Integer): Integer;
C++	int GetControllerTWDelay (int *plCtrTWDelay);
LabView	
Parameter	CtrTWDelay: Regler-Verzögerung [ms]
Beispiel	LS.GetControllerTWDelay(&CtrTWDelay);

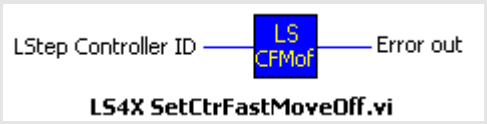
LS_SetControllerTWDelay	
Beschreibung	Regler-Verzögerung
Delphi	function LS_SetControllerTWDelay(CtrTWDelay: Integer): Integer; function LSX_SetControllerTWDelay(LSID: Integer; CtrTWDelay: Integer): Integer;
C++	int SetControllerTWDelay (int lCtrTWDelay);
LabView	
Parameter	CtrTWDelay: Regler-Verzögerung 0 – 100 [ms]
Beispiel	LS.SetControllerTWDelay(0); // Regler-Verzögerung Aus

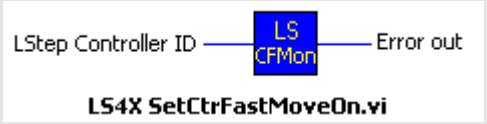
LS_GetCtrFastMove	
Beschreibung:	Liest die Einstellung von der Fast Move Funktion
Delphi:	function LS_ GetCtrFastMoveOff(var bActive: LongBool): Integer; function LSX_ GetCtrFastMoveOff(LSID: Integer; var bActive: LongBool): Integer;
C++:	int GetCtrFastMove (BOOL *pbActive);
LabView:	
Parameter:	bActive: True => Fast Move Funktion aktiv
Beispiel:	LS. GetCtrFastMoveOff (&bActive);

LS_GetCtrFastMoveCounter	
Beschreibung:	Bei einer Reglerdifferenz, die größer als der Fangbereich ist, wird ein neuer Vektor gestartet und der dazu gehörige Counter um eins erhöht. Die Function liefert Fast Move Counters
Delphi:	function LS_ GetCtrFastMoveCounter(var XC, YC, ZC, RC: Integer): Integer; function LSX_ GetCtrFastMoveCounter(LSID: Integer; var XC, YC, ZC, RC: Integer): Integer;
C++:	int GetCtrFastMoveCounter (int *plXC, int *plYC, int *plZC, int *plRC);
LabView:	
Parameter:	XC, YC, ZC, RC: Anzahl ausgeführter Fast Move Funktionen
Beispiel:	LS. SetCtrFastMoveCounter (&XC, &YC, &ZC, &RC);

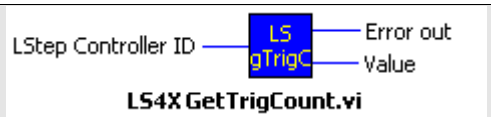
LS_GetTargetWindow	
Beschreibung:	Liest die Zielfenster aller Achsen
Delphi:	function LS_GetTargetWindow(var X, Y, Z, A: Double): Integer; function LSX_GetTargetWindow(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++:	int GetTargetWindow (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter:	X, Y, Z und A: Zielfenster , abhängig von der Dimension
Beispiel:	LS.GetTargetWindow(&X, &Y, &Z, &A);

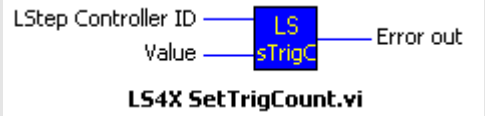
LS_SetTargetWindow	
Beschreibung:	Zielfenster
Delphi:	function LS_SetTargetWindow(X, Y, Z, A: Double): Integer; function LSX_SetTargetWindow(LSID: Integer; X, Y, Z, A: Double): Integer;
C++:	int SetTargetWindow (double dX,double dY,double dZ,double dA);
LabView:	
Parameter:	X, Y, Z und A 1 – 25000 (Motorincremente) 0.1 – Spindelsteigung/2 (µm) 0.0001 – Spindelsteigung/2 (mm) (Werte sind abhängig von der Dimension)
Beispiel:	LS.SetTargetWindow(1.0, 0.002, 1.0, 1.0);

LS_SetCtrFastMoveOff	
Beschreibung:	Fast Move Funktion deaktivieren
Delphi:	function LS_ SetCtrFastMoveOff: Integer; function LSX_ SetCtrFastMoveOff(LSID: Integer): Integer;
C++:	int SetCtrFastMoveOff ();
LabView:	
Parameter:	
Beispiel:	LS. SetCtrFastMoveOff ();

LS_SetCtrFastMoveOn	
Beschreibung:	Fast Move Funktion aktivieren, d. h. bei einer Reglerdifferenz, die größer als der Fangbereich ist, wird ein neuer Vektor gestartet.
Delphi:	function LS_ SetCtrFastMoveOn: Integer; function LSX_ SetCtrFastMoveOn(LSID: Integer): Integer;
C++:	int SetCtrFastMoveOn ();
LabView:	
Parameter:	
Beispiel:	LS. SetCtrFastMoveOn();

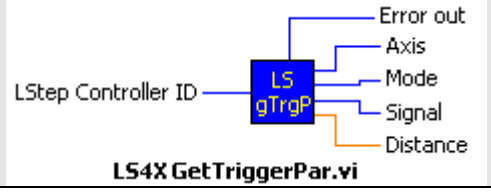
Trigger-Ausgang

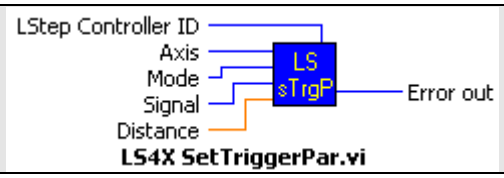
LS_GetTrigCount	
Beschreibung:	Triggerzählerstand lesen.
Delphi:	function LS_GetTrigCount (var Value: Integer): Integer; function LSX_GetTrigCount (LSID: Integer; var Value: Integer): Integer;
C++:	int GetTrigCount (int *pValue);
LabView:	
Parameter:	Value: Anzahl der ausgeführten Trigger
Beispiel:	LS.GetTrigCount (&Value);

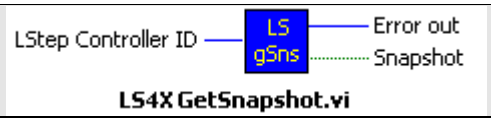
LS_SetTrigCount	
Beschreibung:	Triggerzählerstand setzen.
Delphi:	function LS_ SetTrigCount (Value: Integer): Integer; function LSX_ SetTrigCount (LSID: Integer; Value: Integer): Integer;
C++:	int SetTrigCount (int Wert);
LabView:	
Parameter:	Value: 0 bis 2147483647
Beispiel:	LS.SetTrigCount (0);

LS_GetTrigger	
Beschreibung:	Liefert den aktuellen Trigger-Zustand
Delphi:	function LS_GetTrigger(var ATrigger: LongBool): Integer; function LSX_GetTrigger(LSID: Integer; var ATrigger: LongBool): Integer;
C++:	int GetTrigger (BOOL *pbATrigger);
LabView:	
Parameter:	ATrigger: True => Trigger „Ein“ False => Trigger „Aus“
Beispiel:	LS.GetTrigger(&ATrigger);

LS_SetTrigger	
Beschreibung:	Trigger Ein/Aus
Delphi:	function LS_SetTrigger(ATrigger: LongBool): Integer; function LSX_SetTrigger(LSID: Integer; ATrigger: LongBool): Integer;
C++:	int SetTrigger (BOOL bATrigger);
LabView:	
Parameter:	ATrigger: Trigger Ein/Aus
Beispiel:	LS.SetTrigger(true);

LS_GetTriggerPar	
Beschreibung:	Liefert Trigger-Parameter
Delphi:	function LS_GetTriggerPar(var Axis, Mode, Signal: Integer; var Distance: Double): Integer; function LSX_GetTriggerPar(LSID: Integer; var Axis, Mode, Signal: Integer; var Distance: Double): Integer;
C++:	int GetTriggerPar (int *plAxis, int *plMode, int *plSignal, double *pdDistance);
LabView:	
Parameter:	Axis: Achse (1..4) Mode: Trigger Modus (siehe Befehl !trigm) Signal: Trigger Signal (siehe Befehl !trigs) Distance: Trigger Distanz (siehe Befehl !trigd)
Beispiel:	LS.GetTriggerPar(&Axis, & Mode, & Signal, & Distance);

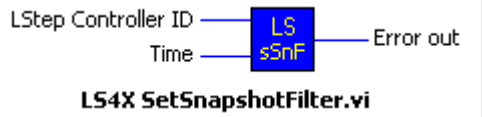
LS_SetTriggerPar	
Beschreibung:	Trigger-Parameter
Delphi:	function LS_SetTriggerPar(Axis, Mode, Signal: Integer; Distance: Double): Integer; function LSX_SetTriggerPar(LSID: Integer; Axis, Mode, Signal: Integer; Distance: Double): Integer;
C++:	int SetTriggerPar (int lAxis, int lMode, int lSignal, double dDistance);
LabView:	
Parameter:	Axis: Achse (1..4) Mode: Trigger Modus (siehe Befehl !trigm) Signal: Trigger Signal (siehe Befehl !trigs) Distance: Trigger Distanz (siehe Befehl !trigd)
Beispiel:	LS.SetTriggerPar(1, 3, 2, 5.0);

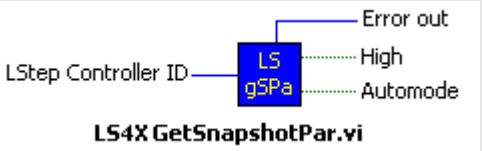
LS_GetSnapshot	
Beschreibung:	Liefert den aktuellen Snapshot-Zustand
Delphi:	function LS_GetSnapshot(var ASnapshot: LongBool): Integer; function LSX_GetSnapshot(LSID: Integer; var ASnapshot: LongBool): Integer;
C++:	int GetSnapshot (BOOL *pbASnapshot);
LabView:	
Parameter:	ASnapshot: True => Snapshot „Ein“ False => Snapshot „Aus“
Beispiel:	LS.GetSnapshot(&ASnapshot);

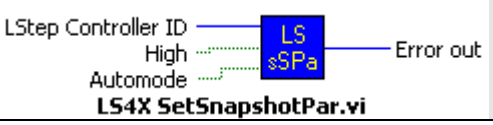
LS_SetSnapshot	
Beschreibung:	Snapshot Ein/Aus
Delphi:	function LS_SetSnapshot(ASnapshot: LongBool): Integer; function LSX_SetSnapshot(LSID: Integer; ASnapshot: LongBool): Integer;
C++:	int SetSnapshot (BOOL bASnapshot);
LabView:	
Parameter:	ASnapshot: Snapshot Ein/Aus
Beispiel:	LS.SetSnapshot(true);

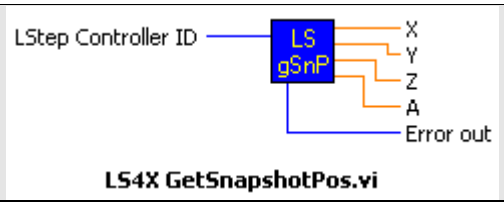
LS_GetSnapshotCount	
Beschreibung:	Snapshot-Zähler
Delphi:	function LS_GetSnapshotCount(var SnsCount: Integer): Integer; function LSX_GetSnapshotCount(LSID: Integer; var SnsCount: Integer): Integer;
C++:	int GetSnapshotCount (int *plSnsCount);
LabView:	
Parameter:	SnsCount: Snapshot-Zähler
Beispiel:	LS.GetsnapshotCount(&SnsCount);

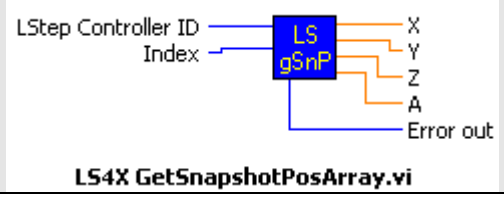
LS_GetSnapshotFilter	
Beschreibung	Eingangsfilter auslesen
Delphi:	function LS_GetSnapshotFilter(var lTime: Integer): Integer; function LSX_GetSnapshotFilter(LSID: Integer; var lTime: Integer): Integer;
C++:	int GetSnapshotFilter (int *plTime);
LabView:	
Parameter:	lTime: Filterzeit [ms]
Beispiel:	LS. GetSnapshotFilter(&lTime);

LS_SetSnapshotFilter	
Beschreibung	Eingangsfilter bei prellenden Schaltern setzen
Delphi:	function LS_SetSnapshotFilter(lTime: Integer): Integer; function LSX_ SetSnapshotFilter(LSID: Integer; lTime: Integer): Integer;
C++:	int SetSnapshotFilter (int lTime);
LabView:	
Parameter:	lTime: Filterzeit, Wertebereich 0 – 100 ms
Beispiel:	LS. SetSnapshotFilter(0); // Kein Eingangsfilter

LS_GetSnapshotPar	
Beschreibung	Snapshot-Parameter auslesen
Delphi:	function LS_GetSnapshotPar(var High, AutoMode: LongBool): Integer; function LSX_GetSnapshotPar(LSID: Integer; var High, AutoMode: LongBool): Integer;
C++:	int GetSnapshotPar (BOOL *pbHigh, BOOL *pbAutoMode);
LabView:	
Parameter:	High: True => Snapshot ist High-Aktiv False => Low-Aktiv AutoMode: True => Snapshot „Automatik“. Die Position wird nach dem ersten Impuls automatisch angefahren.
Beispiel:	LS.GetSnapshotPar(&High, & AutoMode);

LS_SetSnapshotPar	
Beschreibung	Snapshot-Parameter
Delphi:	function LS_SetSnapshotPar(High, AutoMode: LongBool): Integer; function LSX_SetSnapshotPar(LSID: Integer; High, AutoMode: LongBool): Integer;
C++:	int SetSnapshotPar (BOOL bHigh, BOOL bAutoMode);
LabView:	
Parameter:	High: Snapshot High-Aktiv AutoMode: Snapshot-Position automatisch anfahren
Beispiel:	LS.SetSnapshotPar(true, false);

LS_GetSnapshotPos	
Beschreibung	Snapshot-Position auslesen
Delphi	function LS_GetSnapshotPos(var X, Y, Z, A: Double): Integer; function LSX_GetSnapshotPos(LSID: Integer; var X, Y, Z, A: Double): Integer;
C++	int GetSnapshotPos (double *pdX, double *pdY, double *pdZ, double *pdA);
LabView:	
Parameter	X, Y, Z, A: Positionswerte
Beispiel	double X, Y, Z, A; LS.GetSnapshotPos(&X, &Y, &Z, &A);

LS_GetSnapshotPosArray	
Beschreibung	Snapshot-Position aus Array auslesen
Delphi	function LS_GetSnapshotPosArray(Index: Integer; var X, Y, Z, R: Double): Integer; function LSX_GetSnapshotPosArray(LSID: Integer; Index: Integer; var X, Y, Z, R: Double): Integer;
C++	int GetSnapshotPosArray (int lIndex, double *pdX, double *pdY, double *pdZ, double *pdR);
LabView:	
Parameter	Index: Nummer der Snapshot-Position (1-200) X, Y, Z, A: Positionswerte
Beispiel	double X, Y, Z, A; LS.GetSnapshotPos(2, &X, &Y, &Z, &A);

9.5 Fehlercodes

LStep-Nr	API-Nr	Kommentar
0	0	Kein Fehler
	4001,4002	Interner Fehler
	4003	Undefinierter Fehler
	4004	Unbekannter Schnittstellentyp (kann bei Connect... auftreten)
	4005	Fehler beim Initialisieren der Schnittstelle
	4006	Keine Verbindung zur Steuerung (z.B. wenn SetPitch vor Connect aufgerufen wird)
	4007	Timeout während Lesen von der Schnittstelle
	4008	Fehler bei Befehlsübertragung an die LSTEP
	4009	Befehl wurde abgebrochen (mit SetAbortFlag)
	4010	Befehl wird von API nicht unterstützt
11	4011	Joystick-Hand eingeschaltet (kann bei SetJoystickOn/Off auftreten)
11	4012	Kein Verfahr-Befehl möglich, da Joystick-Hand
	4013	Regler-Timeout
12	4015	Endschalter in Verfahrrichtung betätigt
14	4017	Fehler beim Kalibrieren (Endschalter nicht korrekt freigefahren)
1	4101	Keine gültige Achsenbezeichnung
2	4102	Keine ausführbare Funktion
3	4103	Zu viele Zeichen in Befehls-String
4	4104	Kein gültiger Befehl
5	4105	Außerhalb des gültigen Zahlenbereiches
6	4106	Falsche Anzahl der Parameter
7	4107	Kein !oder ?
8	4108	Kein TVR möglich, da Achse aktiv
9	4109	Kein Ein- oder Ausschalten der Achsen, da TVR aktiv
10	4110	Funktion nicht konfiguriert
11	4111	Kein Move-Befehl möglich, da Joystick-Hand
12	4112	Endschalter betätigt
13	4113	Function kann nicht ausgeführt werden, da Encoder erkannt (clear pos)
14	4114	Fehler beim Kalibrieren (Endschalter nicht korrekt freigefahren)
15	4115	Wird beim Freifahren des Endschalters beim Kalibrieren oder Tischhubmessen der gegenüberliegende Endschalter aktiv, wird diese Funktion abgebrochen.
20	4120	Treiberrelaise defekt (Sicherheitskreis K3/K4)
21	4121	Es dürfen nur einzelne Vektoren verfahren werden (Einrichtbetrieb)
22	4122	Es darf kein Kalibrieren, Tischhubmessen oder Joystick-Betrieb durchgeführt werden (Tür offen oder Einrichtbetrieb)
23	4123	SECURITY Error X-Achse
24	4124	SECURITY Error Y-Achse
25	4125	SECURITY Error Z-Achse
26	4126	SECURITY Error A-Achse
27	4127	Stop aktiv
28	4128	Fehler im Türschaltersicherheitskreis (nur bei LS44/Solero)
29	4129	Endstufen nicht eingeschaltet (nur bei LS44)
30	4130	GAL Sicherheitsfehler (nur bei LS44)
31	4131	Joystick lässt sich nicht einschalten, da Move aktiv

9.6 Häufige Fragen & Antworten

Wie werden die LSTEP4.DLL bzw. die LSTEP4X.DLL in einem MS Visual C++ Projekt eingebunden?
Wie initialisiere ich mit dem LStep API die Verbindung zur LStep? Welcher der Connect-Befehle sollte verwendet werden?
Wie installiere ich den Treiber für die LStep-PCI?
Warum bekommt mein Programm mit der LSTEP4.DLL keine Verbindung zur LStep-PCI?
Im Ablauf, in der LSTEP4.DLL oder in meinem Programm ist ein Fehler aufgetreten. Wo liegt die Ursache, und wie lässt sich das Problem lösen?
Kann während Verfahrbefehlen der Status von Eingängen, die aktuelle Position u.ä. abgefragt werden?
Warum werden während der Ausführung von LSTEP-API-Funktionen Messages verarbeitet, und wie kann man dies deaktivieren?
Wann sind Moves mit bzw. ohne Wait zu verwenden?
Wie kann ich mit dem LSTEP-API einzelne Achsen der LStep unabhängig voneinander verfahren?
Wie kann ich mehrere LStep-PCI-Karten in einem PC verwenden?
Wann sollte die LSTEP4.DLL, wann die LSTEP4X.DLL verwendet werden?
Ist das LSTEP-API kompatibel zur MCL bzw. zum alten Register-Befehlssatz?
Warum bekomme ich in MS Visual C++ bei Einbindung von LStep4.cpp die Meldung “fatal error C1010“?
Wie kann ich einen speziellen/neuen LStep-Befehl verwenden, für den es keine passende LSTEP-API-Funktion gibt?
Warum sehe ich im Debugger meiner Entwicklungsumgebung bei Verwendung des LSTEP-API die Meldung „First chance exception“, „Exception: Timeout read RS232!“ o.ä.?
Wie kann ich mit dem LSTEP-API eine Art Joystick realisieren, also eine Achse solange fahren, bis eine Taste wieder losgelassen wird?
Wie kann ich die Einstellungen der LStep dauerhaft speichern?
Wie viele Einträge passen in das Protokollfenster des LStep-APIs?

Wie werden die LSTEP4.DLL bzw. die LSTEP4X.DLL in einem MS Visual C++ Projekt eingebunden?

- Projekt erzeugen
- LSTEP4.DLL, LSTEP4.h, LSTEP4.cpp in Projektordner kopieren
- LSTEP4.h und LSTEP4.cpp in Projekt einfügen
- Menü: Projekt\ Einstellungen\ C/C++ Option: [vorkompilierte Header nicht verwenden] wählen
- in LSTEP4.h #include „stdafx.h“ einbinden
- in Projektname_Dlg.h #include „LSTEP4.h“
- die erforderliche Instanz in public einbinden
  Beispiel: CLStep* MyLStep = new CLStep();

Die Einbindung der LSTEP4X.DLL erfolgt analog zu dieser Vorgehensweise.

Wie initialisiere ich mit dem LStep API die Verbindung zur LStep?

Welcher der Connect-Befehle sollte verwendet werden?

Die Verbindung zur LSTEP-API wird mit einem der Connect-Befehle (Connect, ConnectEx, ConnectSimple) initialisiert. Abgesehen von einigen Sonderfällen sollte immer **ConnectSimple** verwendet werden.

ConnectSimple	bei der direkten Übergabe der Schnittstellenparameter
Connect	nach zuvorigem Laden der Schnittstellenparameter aus einer .ini Datei mittels LoadConfig
ConnectEx	beim Laden der Schnittstellenparameter aus einer Datenstruktur

Wie installiere ich den Treiber für die LStep-PCI?

Nach korrektem Einbau der LStep-PCI fordert Windows beim Start einen Treiber für ein Gerät des Typs „Netzwerkcontroller“ an. In diesem Dialog-Fenster klicken Sie auf den Button „Durchsuchen“ o.ä. und wechseln dann in das Verzeichnis, in welches die Dateien des LSTEP-API entpackt wurden. Im Unterordner „LStepPCI“ sind die Treiber-Dateien und die Inf-Dateien, welche für die Treiber-Installation benötigt werden, enthalten.

Warum bekommt mein Programm mit der LSTEP4.DLL keine Verbindung zur LStep-PCI?

Sie sollten zunächst im Windows-Geräte-Manager überprüfen, ob dort die eingebaute LStep-PCI als Gerät eingetragen ist. Außerdem **muss die Datei DRVX40.DLL im Verzeichnis der LSTEP4.DLL, ihres Programms oder einem Windows-Systemordner liegen.** Sie finden diese Datei im Unterordner „LStepPCI“ des LStep API. *(siehe auch Kapitel 9.7).*

Im Ablauf, in der LSTEP4.DLL oder in meinem Programm ist ein Fehler aufgetreten. Wo liegt die Ursache, und wie lässt sich das Problem lösen?

Damit eine Fehlerdiagnose möglich ist, sollten sie unbedingt die Protokollierung des LSTEP-API mit SetWriteLogText einschalten. Anschließend sollten Sie versuchen, den aufgetretenen Fehler bei laufender Protokollierung zu reproduzieren. Die Log-Datei (LStep4.log) können Sie dann zur Analyse an uns mailen.

Kann während Verfahrbefehlen der Status von Eingängen, die aktuelle Position u.ä. abgefragt werden?

Ja, zum Beispiel indem man über einen Windows-Timer oder einen zweiten Thread Funktionen wie GetPos, GetDigitalInputs während eines Verfahrbefehls aufruft. Es ist aber nicht möglich, während eines Verfahrbefehls mit Wait=true den Befehl WaitForAxisStop aufzurufen.

Warum werden während der Ausführung von LSTEP-API-Funktionen Messages verarbeitet, und wie kann man dies deaktivieren?

Das LSTEP-API verarbeitet beim Warten auf Rückmeldungen von Verfahrbefehlen Botschaften, damit das Programm nicht „steht“, denn ansonsten wäre es nicht möglich, in Fehlersituationen einen Abbruch durchzuführen bzw. die Achsen zu stoppen. SetProcessMessagesProc ermöglicht das Ersetzen der internen Message-Dispatching Prozedur des LSTEP-API. Das LSTEP-API verarbeitet während des Wartens auf Rückmeldungen der LStep im Main-Thread Messages. Wenn sie das Message-Dispatching abschalten wollen oder durch eigenen Code ersetzen wollen, können sie SetProcessMessagesProc zum Setzen einer Callback-Prozedur verwenden.

Wann sind Moves mit bzw. ohne Wait zu verwenden?

Move-Befehle mit WaitForAxisStop sind zu verwenden, wenn alle Achsen synchronund linear interpoliert verfahren werden sollen. Die Steuerung nimmt neue Move-Befehle erst entgegen, wenn alle Achsen stehen.

Move-Befehle ohne WaitForAxisStop sind zu verwenden, wenn die Achsen asynchron verfahren werden sollen. Der Anwender hat in diesem Fall dafür zu sorgen, dass nur diejenige Achse die steht auch einen neuen Move-Befehl bekommt.

Wie kann ich mit dem LSTEP-API einzelne Achsen der LStep unabhängig voneinander verfahren?

Die Verfahr-Befehle des LSTEP-API bieten zwei verschiedene Möglichkeiten: Wird der (letzte) Parameter Wait=true gesetzt, kehrt die Funktion erst zurück, nachdem die Achsen ihre Zielposition erreicht haben. Wird hingegen dieser **Parameter Wait=false** gesetzt, sendet die LSTEP-API-Funktion nur den Verfahrbefehl und kehrt unmittelbar zurück, ohne auf die Ausführung der Bewegung zu warten.
Indem man also zunächst MoveAbsSingleAxis mit Wait=false für die X-Achse verwendet und etwas später MoveAbsSingleAxis mit Wait=false für die Y-Achse aufruft, können Achsen separat verfahren werden. Um festzustellen, ob Achsen ihre Zielposition erreicht haben, kann man den Befehl **WaitForAxisStop** benutzen.

Beispiel:

```
LS.MoveAbsSingleAxis(Xaxis, 10, false);   // Verfahre die X-Achse asynchron
Delay(1000);                              // Warte 1s bis zum Start der Y-Achse
LS.MoveAbsSingleAxis(Yaxis, 20, false);   // Verfahre die Y-Achse asynchron
LS.WaitForAxisStop(3, 0, flag);           // Warten bis X- und Y-Achse gestoppt
                                          haben, ohne Timeout
```

Es ist aber **nicht möglich, Move-Befehle mit Wait=true und solche mit Wait=false gleichzeitig zu verwenden**. Dies führt zu permanenten oder sporadischen Fehlern in der Kommunikation.

Beispiel:

Nicht erlaubt:

```
LS.MoveAbsSingleAxis(Xaxis, 10, false);   // Verfahre die X-Achse asynchron
LS.MoveAbsSingleAxis(Yaxis, 20, true);    // Verfahre die Y-Achse asynchron ohne
                                          auf das Ende des asynchronen
                                          Verfahrbefehls gewartet zu haben
```

Wie kann ich mehrere LStep-PCI-Karten in einem PC verwenden?

Die Vorgehensweise bei der Installation ist diese wie bei einer einzelnen Karte. Nach dem Start fordert Windows den Treiber für sämtliche LStep-PCI-Karten an.
Doch es ist **problematisch festzustellen, welche physikalische Karte zu einer bestimmten Index-Nummer gehört**. Es ist nicht sichergestellt, dass man durch LS_ConnectSimple(4, nil, 0, true) eine Verbindung zur LStep-PCI im ersten PCI-Slot des Mainboards, durch LS_ConnectSimple(4, nil, 1, true) eine Verbindung zur LStep-PCI im zweiten PCI-Slot erhält etc. Deshalb sollte zur eindeutigen Identifikation der Karten die Seriennummer mit **GetSerialNr** abgefragt werden.

Wann sollte die LSTEP4.DLL, wann die LSTEP4X.DLL verwendet werden?

Wenn mehrere LSteps/LStep-PCI-Karten von einem PC aus gesteuert werden sollen, sollte die LSTEP4X.DLL eingesetzt werden, ansonsten ist die LSTEP4.DLL geeignet.

Ist das LSTEP-API kompatibel zur MCL bzw. zum alten Register-Befehlssatz?

Das LSTEP-API ist prinzipiell abwärtskompatibel zu dem Register-Befehlssatz, mit dem die MCL und ältere LSteps kommunizieren. Jedoch bietet dieser Befehlssatz viele Möglichkeiten nicht, die das LStep API bei Steuerungen mit neuem Befehlssatz verwenden kann. Deshalb können einige LSTEP-API-Befehle wie WaitForAxisStop bei Steuerungen mit altem Befehlssatz generell nicht verwendet werden.

Warum bekomme ich in MS Visual C++ bei Einbindung von LStep4.cpp die Meldung "fatal error C1010"?

Es handelt sich hierbei nicht um einen Fehler in der Datei LStep4.cpp. Die Meldung tritt gewöhnlich auf, wenn der Compiler nach der vorkompilierten Header-Datei sucht und sie nicht findet. Sollte in MS Visual C++ die Meldung „fatal error C1010 precompiled header files" auftreten, muss die Option „vorkompilierte Header-Datei" für LStep4.cpp abgeschaltet werden. Sofern Sie die MFC in Ihrem Projekt nicht verwenden, sollten Sie die Zeile #include "stdafx.h" aus LStep4.cpp entfernen.

Wie kann ich einen speziellen/neuen LStep-Befehl verwenden, für den es keine passende LSTEP-API-Funktion gibt?

Die LSTEP-API-Funktion **SendString** bietet die Möglichkeit, um neue, nicht im LSTEP-API vorgesehene LStep-Befehle zu benutzen. Zu beachten ist, dass alle Befehle mit dem Zeichen #13 bzw \r abschließen!

Warum sehe ich im Debugger meiner Entwicklungsumgebung bei Verwendung des LSTEP-API die Meldung „First chance exception", „Exception: Timeout read RS232!" o.ä.?

Interne Exceptions der LSTEP4.DLL, die nur im Debugger sichtbar sind haben keine Bedeutung. Sie dienen zur internen Ablaufsteuerung. Bei ConnectSimple tritt häufig eine Exception auf, da das LSTEP-API versucht, den Befehlssatz herauszufinden. Dabei kommt es zu einem Timeout, wenn die Steuerung den getesteten Befehlssatz nicht unterstützt. In Delphi können Sie in den Debugger-Optionen die entsprechenden Exception zu den von Debugger zu ignorierenden Exceptions hinzufügen.

Wie kann ich mit dem LSTEP-API eine Art Joystick realisieren, also eine Achse solange fahren, bis eine Taste wieder losgelassen wird?

Ein solcher Tasten-Joystick kann folgendermaßen implementiert werden:
Bei Tastendruck die Achse mit einem sehr langen Vektor starten
MoveRelSingleAxis(Xaxis, 100000, false)
Wichtig ist, den Parameter Wait=false zu setzen
Bei Loslassen der Taste den Befehl **StopAxes** aufrufen

Wie kann ich die Einstellungen der LStep dauerhaft speichern?

Der LSTEP-API-Befehl **LstepSave** kann eingesetzt werden, um einmal gemachte Einstellungen (Spindelsteigungen, Getriebefaktoren, Achsenströme usw.) auch nach Reset der LStep zu erhalten. Ob Ihre LStep diesen Befehl unterstützt, können Sie der Dokumentation entnehmen.

Wie viele Einträge passen in das Protokollfenster des LStep-APIs?

Das Protokollfenster des LStep-APIs kann 20.000 Einträge fassen. Treten mehr Einträge auf, so werden die ältesten überschrieben.

Wie viele Einträge können in die Log- Datei geschrieben werden?

In die Log- Datei wird so lange geschrieben, bis das Programm beendet wird oder die Festplatte voll ist.

9.7 Verwendung der LStep PCI-Karte

Das LStep API (LSTEP4.DLL und LSTEP4X.DLL) unterstützt ab der Version 1.0.7.0 die PCI-Einsteckkarte **LStep-PCI** unter den Betriebssystemen Windows 95, 98, NT 4.0 und 2000.

Zum Betrieb wird ein Treiber benötigt, der sich im Unterverzeichnis ‚LStepPCI' befindet. Unter Windows 9x/2000 muss zur Installation des Treibers nach der automatischen Erkennung der LStep-PCI durch das Betriebssystem die .INF-Datei LSPCIW9X.INF bzw. LSPCIW2K.INF ausgewählt werden.
Die Installation des Treibers unter auf Windows NT basierenden Betriebssystemen (Windows NT, Windows 2000, Windows XP) kann mit Hilfe des Tools SetupDrvXNT.exe (ab API-Version 1.2.0. 20 SetupDrvX.exe) durchgeführt werden. Hierfür ist diese Datei nach der Installation der Karte einmal auszuführen.

Beispiel zur Initialisierung des LStep API mit einer LStep PCI:
LS_ConnectSimple(4, nil, 0, true);
(4 = LS_if_PCI)
Der dritte Parameter gibt den Index der Karte an. Sind in einem PC mehrere LStep-PCI-Karten installiert, so werden diese von 0 bis n-1 nummeriert.

Abgesehen von der Initialisierung mit LS_ConnectSimple unterscheiden sich die Funktionsaufrufe des LStep API nicht von denen, die bei einer normalen LStep verwendet werden, sodaß ein und dasselbe Programm bei Einsatz des LStep API mit minimalen Änderungen im Quellcode sowohl eine LStep über RS232 als auch eine LStep-PCI ansteuern kann.

Die Datei DRVX40.DLL (befindet sich im Unterverzeichnis ‚LStepPCI') sollte im selben Verzeichnis wie die LSTEP4.DLL liegen, damit die LStep-PCI verwendet werden kann. (Oder in einem in der Umgebungsvariable PATH angebenen Verzeichnis)

9.7.1 Interrupt-gesteuerte Kommunikation mit LStep-PCI

Die LStep4.Dll (LStep4X.Dll) kommuniziert ab der Version 1.2.0.20 mit der LStep-PCI Karte per Interrupt. Dies erhöht die Durchsatzrate von Move und Status-Kommandos erheblich.

Es wird nicht empfohlen 2 LStep-PCI Karten in die Slots zu stecken, welche sich eine Interrupt Request Line (IRQ) teilen. Dies läßt sich im Hardwaremanager überprüfen. (*Start => Einstellungen => Systemsteuerung => System => Hardware => Gerätemanager => Lstep => LStep-PCI => Resourcen)*

Teilen sich 2 LStep-PCI Karten eine Interrupt Request Line und soll mit beiden Karten gleichzeitig kommuniziert werden, dann wird die Lstep4X.Dll mit einer von den beiden Karten per Pollen kommunizieren. Dadurch wird diese Kommunikation etwas langsamer.

Arbeitet eine LStep-PCI Karte noch mit der alten Firmware, wird die neue Lstep4.DLL (Lstep4X.Dll) per Pollen kommunizieren.
Ab der Firmware-Version 38 / interne Version T02.21.05 ist die Kommunikation Interrupgesteuert

Nach der Treiberinstallation muß die neue SetupDriverX.exe ausgeführt werden. Damit wird der Treiber so konfiguriert, daß die interruptgesteuerte Kommunikation möglich wird.

Bitte beachten sie die Readme.txt Datei !

9.7.2 Readme

Installation der Treiber für LStep-

Windows NT
1) Die Dateien DRIVERX.SYS, DRVX40.DLL, SetupDrvX.exe in einen Ordner kopieren.
2) SetupDrvX.exe starten.

Windows 9x
1) DRIVERX.VXD, DRVX40.DLL, LSPCIW9X.INF, SetupDrvX.exe in einen Ordner kopieren.
2) Mit dem Hardware-Assistenten den Treiber installieren.
3) SetupDrvX.exe starten, um den Treiber zu konfigurieren.

Es spielt keine Rolle, ob man Schritt 2) vor 3) durchführt oder danach.
Für X LStep-PCI-Karten muß Schritt 2) X mal durchgeführt werden.

Windows 2000, XP
1) DRVXWDM.SYS, DRVX40.DLL, LSPCIW2K.INF, SetupDrvX.exe in einen Ordner kopieren.
2) Mit dem Hardware-Assistenten den Treiber installieren.
3) SetupDrvX.exe starten, um den Treiber zu konfigurieren.

Es spielt keine Rolle, ob man Schritt 2) vor 3) durchführt oder danach.
Für X LStep-PCI-Karten muß Schritt 2) X mal durchgeführt werden.

9.7.3 API / LSTEP Befehle

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
Connect	Mit LSTEP verbinden	-
ConnectEx	Mit LSTEP verbinden	-
ConnectSimple	Mit LSTEP verbinden	-
CreateLSID	Erzeugt eine ID Nr bei der Verwendung des LSTEP4X APIs	-
Disconnect	Verbindung zu LSTEP trennen	-
EnableCommandRetry	Mit dieser Funktion kann das wiederholte Senden von Kommandos im Falle von Fehlern ein-/ausgeschaltet werden	-
FlushBuffer	Löscht den Eingabepuffer	-
FreeLSID	Gibt die erzeugte ID Nr wieder frei	-
LoadConfig	LSTEP-Konfiguration (Schnittstelle, Achseinstellungen, Regler) aus INI-Datei laden	-
SaveConfig	LSTEP-Konfiguration (Schnittstelle, Achseinstellungen, Regler) in INI-Datei speichern.	-
SendString	String an LSTEP senden	-
SendStringPosCmd	Verfahrbefehl, welcher Rückmeldung erwartet, als String an LSTEP senden	-
SetAbortFlag	Flag setzen, damit die Kommunikation mit der LSTEP abgebrochen wird	-
SetControlPars	Überträgt die mit LS_LoadConfig geladenen Parameter an die LSTEP	-
SetCorrTblOff	Achsenkorrektur deaktivieren	-
SetCorrTblOn	Achsenkorrektur in x/y-Matrix mit linearer Interpolation aktivieren	-
SetExtValue	Erweiterungen einschalten	-
SetFactorMode	Positionswert-Umrechnung für ‚krumme' Spindelsteigungen	-
SetLanguage	Sprachumschaltung LSTEP-API (Protokoll/Meldungen)	-
SetProcessMessagesProc	Ermöglicht das Ersetzen der internen Message-Dispatching Prozedur des LStep API	-
SetShowCmdList	LStep-API Befehlsliste Ein/Aus	-
SetShowProt	Schnittstellen-Protokoll Ein/Aus	-
SetWriteLogText	Schreiben der Protokoll-Datei LSTEP4.log ein-/ausschalten (Standardmäßig ist das Schreiben in LSTEP4.log ausgeschaltet)	-
SetWriteLogTextFN	Schreiben des Schnittstellen-Protokolls in eine bestimmte Datei ein-/ausschalten	-

Steuerungs-Info:

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
GetSerialNr	Seriennummer der Steuerung auslesen	?readsn
GetVersionStr	liefert die aktuelle Versionsnummer der Firmware zurück	?ver
GetVersionStrDet	Detaillierte Versionsnummer der Firmware auslesen	?det
GetVersionStrInfo	Ergänzung zur aktuellen Versionsnummer auslesen	?iver

Einstellungen

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
GetAccel	Beschleunigung abfragen	?accel
GetActiveAxes	Liefert die Achsenfreigaben	?axis
GetAxisDirection	Drehrichtungs-Umkehr abfragen	?axisdir
GetCalibBackSpeed	Liefert die Geschwindigkeit, mit der aus den Endschaltern gefahren wird	?calbspeed
GetCaliboffset	Kalibrier-Offset abfragen	?caliboffset
GetCalibrateDir	Liefert Vorzeichen-Umkehr bei Kalibration	?caldir
GetCurrentDelay	Gibt an Zeitverzögerung für die Stromabsenkung	?curdelay
GetDimensions	Abfrage der Dimensionen der Achsen	?dim
GetGear	Getriebefaktor abfragen	?gear
GetJoystickFilter	Gibt an, ob Filterung im Joystick-Betrieb aktiv ist	?joyfilter
GetMotorCurrent	Motorstrom abfragen	?cur
GetMotorTablePatch	Gibt an, ob die Korrekturtabelle aktiviert ist	?mtpatch
GetOutFuncLev	Liefert die StromUmschaltungsgeschwindigkeit	?opfl
GetPitch	Liefert die Spindelsteigungen	?pitch
GetPowerAmplifier	Endstufen Ein/Aus (nur bei LS44)	?pa
GetReduction	Stromabsenkung abfragen	?reduction
GetRefSpeed	Geschwindigkeit, mit der beim Kalibrieren die Referenzmarke gesucht wird, abfragen.	?calrefspeed
GetRMOffset	RM-Offset abfragen	?rmoffset
GetSpeedPoti	Gibt an, ob Speed-Poti Ein oder Aus ist.	?pot
GetStopAccel	Liefert die Bremsbeschleunigung, wenn der Stopeingang aktiv ist	?stopaccel
GetStopPolarity	Stopeingang Polarität lesen	?stoppol
GetVel	Geschwindigkeit aller Achsen abfragen	?vel
GetVelFac	Geschwindigkeitsuntersetzung abfragen	?velfac
GetVLevel	Liefert die Ausgeblendete Geschwindigkeit	?vlevel
GetXYAxisComp	Abfrage der XY-Achsüberlagerung	?xycomp
LstepSave	Aktuelle Konfiguration in LStep speichern (EEPROM)	save
SetAccel	Beschleunigung einstellen	!accel
SetAccelSingleAxis	Beschleunigung für eine einstellen	!accel x (y,z,a)
SetActiveAxes	Achsenfreigabe	!axis
SetAxisDirection	Drehrichtungs-Umkehr	!axisdir
SetCalibBackSpeed	Geschwindigkeit, mit der aus den Endschalter gefahren wird, setzen	!calbspeed
SetCaliboffset	Kalibrier-Offset	!caliboffset
SetCalibrateDir	Vorzeichen-Umkehr bei Kalibration	!caldir
SetCurrentDelay	Zeitverzögerung für die Stromabsenkung	!curdelay
SetDimensions	Dimensionen der Achsen einstellen	!dim
SetGear	Getriebefaktor einstellen	!gear
SetJoystickFilter	Filterung im Joystick-Betrieb Ein/Aus	!joyfilter
SetMotorCurrent	Motorstrom einstellen	!cur
SetMotorTablePatch	Korrekturtabelle Ein/Aus	!mtpatch
SetOutFuncLev	Setzt die StromUmschaltungsgeschwindigkeit	!opfl
SetPitch	Spindelsteigung setzen	!pitch
SetPowerAmplifier	Schaltet bei LS44 Endstufen Ein/Aus	!pa
SetReduction	Stromabsenkung einstellen	!reduction
SetRefSpeed	Geschwindigkeit, mit der beim Kalibrieren die Referenzmarke gesucht wird, setzen	!calrefspeed
SetRMOffset	RM-Offset	!rmoffset

SetSpeedPoti	Speed-Poti Ein/ Aus	!pot
SetStopAccel	Stellt die Bremsbeschleunigung ein, wenn der Stopeingang aktiv ist	!stopaccel
SetStopPolarity	Stopeingang Polarität einstellen	!stoppol
SetVel	Geschwindigkeit aller Achsen einstellen	!vel
SetVelFac	Geschwindigkeitsuntersetzung setzen	!velfac
SetVelSingleAxis	Geschwindigkeit für eine Achse einstellen	!vel x (y,z,a)
SetVLevel	Ausblenden von Geschwindigkeiten, bei denen Resonanzen auftreten	!vlevel
SetXYAxisComp	XY-Achsüberlagerung aktivieren	!xycomp
SoftwareReset	Software wird in den Startzustand versetzt	reset

Statusabfragen

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
GetError	liefert die aktuelle Fehlernummer	?err
GetSecurityErr	Liest alle Zustände und Ergebnisse der GAL-Sicherheitsüberwachung (nur bei LS44-Steuerungen)	?securityerror
GetSecurityStatus	Liest den aktuellen Zustand der Sicherheitsüberwachung	?securitystatus
GetStatus	liefert den aktuellen Zustand der Steuerung	?status
GetStatusAxis	liefert den aktuellen Zustand der einzelnen Achsen	?statusaxis
GetStatusLimit	Liefert den aktuellen Zustand der Software-Grenzen jeder einzelnen Achse	?statuslimit
SetAutoStatus	AutoStatus Ein/Aus	!autostatus

Fahrbefehle und Positionsverwaltung

APIBefehl	Kurzbeschreibung	LSTEP-Befehl
Calibrate	Kalibrieren	!cal
CalibrateEx	Es werden nur die Achsen kalibriert, deren entsprechendes Bit in dem übergebenen Integer-Wert gesetzt ist.	!cal x (xy,z,a)
Clearpos	Positionswerte werden genullt (für endlos Drehachsen)	!clearpos
GetDelay	Liefert die Verzögerung des Vektorstarts	?delay
GetDistance	Liefert die Strecke, die mit LS_PosRelShort gestartet wird	?distance
GetPos	Abfrage der aktuellen Position aller Achsen	?pos
GetPosEx	Abfrage der aktuellen Geber- bzw. Positionswerte aller Achsen	
GetPosSingleAxis	Abfrage der aktuellen Position einer Achse	?pos x (y,z,a)
MoveAbs	Absolutposition anfahren	!moa
MoveAbsSingleAxis	Absolutposition einer Achse anfahren	!moa x (y,z,a)
MoveEx	Erweiterter Verfahr-Befehl	
MoveRel	Relativen Vektor fahren	!mor
MoveRelShort	Positionieren Relativ (short command)	m
MoveRelSingleAxis	Relativen Vektor einer Achse verfahren	!mor x (y,z,a)
RMeasure	Tischhub messen	!rm
RmeasureEx	Tischhub messen wird nur bei den Achsen durchgeführt, deren entsprechendes Bit in dem übergebenen Integer-Wert gesetzt ist	!rm x (xy z)

SetDelay	Durch den Befehl Delay kann eine Verzögerung des Vektorstarts erzeugt werden	!delay
SetDistance	Strecke setzen (für MoveRelShort)	!distance
SetPos	Position setzen	!pos
StopAxes	alle Verfahrbewegungen werden abgebrochen	a
WaitForAxisStop	Die Funktion kehrt zurück, sobald die in der Bit-Maske AFlags gewählten Achsen ihre Zielposition erreicht haben	-

Joystick und Handrad

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
GetDigJoySpeed	Digitaler Joystick und Geschwindigkeit lesen	?speed
GetHandwheel	Liest den Zustand des Handrads	?hw
GetJoystick	Liest den Zustand des Analog-Joysticks	?joy
GetJoystickDir	Richtung Joystick	?joydir
GetJoystickWindow	Joystick-Fenster ablesen	?joywindow
SetDigJoySpeed	Digitaler Joystick und Geschwindigkeit setzen	!speed
SetHandwheelOff	Handrad Aus	!hw 0
SetHandwheelOn	Handrad Ein	!hw 1 (1-4)
SetJoystickDir	Richtung Joystick	!joydir
SetJoystickOff	Analog-Joystick Aus	!joy 0
SetJoystickOn	Analog-Joystick Ein	!joy 1 (1-4)
SetJoystickWindow	Joystick-Fenster setzen	!joywindow
GetJoyChangeAxis	Liest Joystickachszuordnung	?joychangeaxis
JoyChangeAxis	Setzt Joystickachszuordnung	!joychangeaxis

Bedienpult mit Trackball und Joyspeed-Tasten

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
GetBPZ	Liest Zustand des Bedienpults	?bpz
GetBPZJoyspeed	Liest Bedienpult Joystick-Speed	?joyspeed
GetBPZTrackballBackLash	Liest Bedienpult Trackball-Umkehrspiel	?bpzbl
GetBPZTrackballFactor	Liest Bedienpult Trackball-Faktor	?bpztf
SetBPZ	Bedienpult Ein/Aus	!bpz
SetBPZJoyspeed	Bedienpult Joystick-Speed	!joyspeed
SetBPZTrackballBackLash	Bedienpult Trackball-Umkehrspiel	!bpzbl
SetBPZTrackballFactor	Bedienpult Trackball-Faktor	!bpztf

Endschalter (Hardware u. Software)

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
GetAutoLimitAfterCalibRM	Gibt an, ob beim Kalibrieren und Tischhubmessendie interne Software-Limits gesetzt werden.	?nosetlimit
GetLimit	Liefert Verfahrbereichsgrenzen	?lim
GetLimitControl	Liest,ob die Bereichsüberwachung eingeschaltet ist	?limctr
GetSwitchActive	Gibt an, ob die Endschalter eingeschaltet sind	?swact
GetSwitches	liest den Zustand aller Endschalter	?readsw
GetSwitchPolarity	Liest Endschalterpolarität	?swpol
SetAutoLimitAfterCalibRM	Verhindert, dass beim Kalibrieren und Tischhubmessen die internen Software-Limits gesetzt werden.	!nosetlimit
SetLimit	Verfahrbereichsgrenzen einstellen	!lim
SetLimitControl	Bereichsüberwachung	!limctr
SetSwitchActive	Liest Status für Endschalter Ein / Aus	!swact

SetSwitchPolarity	Endschalterpolarität einstellen	!swpol

Digitale und analoge Ein.- und Ausgänge

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
GetAnalogInput	Lesen des aktuellen Zustands eines Analogkanals	?anain
GetAnalogInputs2	Lesen der aktuellen Zustände der Analogkanäle PT100, MV und V24	--
GetDigitalInputs	Alle Inputpins lesen	?digin
GetDigitalInputsE	Zusätzliche digitale Eingänge lesen (16-31)	?edigin
SetAnalogOutput	Analogkanal setzen	!anaout
SetDigIO_Distance	Aktivierung eines Ausgang in Abhängigkeit der eingestellten Strecke vor/nach der Zielposition	(digfkt)
SetDigIO_EmergencyStop	Funktion der digitalen Ein-/Ausgänge Zuordnung des Not-Stop-Pins	(digfkt)
SetDigIO_Off	Funktion der digitalen Ein-/Ausgänge Aus	(digfkt)
SetDigIO_Polarity	Einstellung der Polarität	!digfkt 16 0 0
SetDigitalOutput	Digitalen Ausgang setzen	!digout x
SetDigitalOutputs	Digitale Ausgänge setzen (0-15)	!digout 0-15
SetDigitalOutputsE	Zusätzliche digitale Ausgänge setzen (16-31)	!edigout

Takt-Vor/Rück Eingänge

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
GetFactorTVR	Liest den Faktor Takt Vor / Rück	?tvrf
GetTVRMode	Einstellung vom Takt Vor / Rück auslesen	?tvr
SetFactorTVR	Faktor Takt Vor / Rück	!tvrf
SetTVRMode	Takt Vor / Rück einstellen	!tvr (0-4)

Takt-Vor/Rück über Schnittstelle

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
SetTVRInPulse	Takt-Vor/Rück über Schnittstelle	px/nx

Takt-Vor/Rück Ausgänge für weitere Achsen

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
GetAccelTVRO	Alle eingestellten Beschleunigungen lesen	?tvroa
GetPosTVRO	Liefert Positionswerte, in Abhängigkeit von Dimension	?tvropos
GetStatusTVRO	Liefert den aktuellen Status der Achsen	?tvrostatus
GetTVROutMode	Einstellung vom Takt Vor/Rück lesen	?tvrout
GetTVROutPitch	Liest Spindelsteigung	?tvropitch
GetTVROutResolution	Liefert dieAuflösung der an zu steuernden Endstufe	?tvrores
GetVelTVRO	Alle eingestellten Geschwindigkeiten lesen	?tvrov
MoveAbsTVROSingleAxis	Absolutposition einer Achse anfahren	!tvromoa x
MoveAbsTVRO	Absolutposition anfahren	!tvromoa
MoveRelTVROSingleAxis	Relativen Vector einer Achse verfahren	!tvromor x
MoveRelTVRO	Relativen Vector verfahren	!tvromor
SetAccelSingleAxisTVRO	Beschleunigung einer einzelnen Achse setzen	!tvroa x
SetAccelTVRO	Beschleunigungen setzen	!tvroa
SetPosTVRO	Position setzen	!tvropos
SetTVROutMode	Takt Vor / Rück einstellen	!tvrout
SetTVROutPitch	Spindelsteigung setzen	!tvropitch
SetTVROutResolution	Auflösung der an zu steuernden Endstufe	!tvrores
SetVelSingleAxisTVRO	Geschwindigkeit einer einzelnen Achse setzen	!tvrov x
SetVelTVRO	Geschwindigkeiten setzen	!tvrov

Geber-Einstellungen

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
ClearEncoder	Geberposition auf null setzen	!pos 0 0 0 0
GetEncoder	Liest alle Geberpositionen	!encpos1 ?pos
GetEncoderActive	Liest, welche Geber nach der Kalibration aktiv werden	?encmask
GetEncoderMask	Geberzustände auslesen	?enc
GetEncoderPeriod	Geberperiodenlängen auslesen	?encperiod
GetEncoderPosition	Liefert Einstellung von Geberwertanzeige	
GetEncoderRefSignal	Gibt an, ob beim Kalibrieren Referenzsignal von Geber ausgewertet werden soll	?encref
SetEncoderActive	Mit dieser Funktion kann ausgewählt werden, welche Geber nach der Kalibration aktiviert werden sollen	!encmask
SetEncoderPeriod	Geberperiodenlängen einstellen	!encperiod
SetEncoderPosition	Geberwertanzeige Ein/Aus	!encpos 1 !pos 0 0 0
SetEncoderRefSignal	Beim Kalibrieren Referenzsignal von Geber auswerten	!encref

Reglereinstellungen

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
ClearCtrFastMoveCounter	Anzahl ausgeführter FastMove Funktionen auf 0 setzen	!ctrfmc 0
GetController	Regler-Modus auslesen	?ctr
GetControllerCall	Einstellung vom Regleraufruf auslesen	?ctrc
GetControllerFactor	Einstellung vom Reglerfaktor auslesen	?ctrf
GetControllerSteps	Regler-Schritte auslesen	?ctrs
GetControllerTimeout	Liefert die Einstellung vom Regler-Überwachungs-Timeout	?ctrt
GetControllerTWDelay	Reglerverzögerung auslesen	?ctrd
GetCtrFastMove	Einstellung von Fast Move Funktion lesen	?ctrfm
GetCtrFastMoveCounter	Anzahl ausgeführter FastMove Funktionen auf 0 auslesen	?ctrfmc
GetTargetWindow	Liefert Reglerzielfenster	?twi
SetController	Regler-Modus einstellen	!ctr
SetControllerCall	Regleraufruf einstellen	!ctrc
SetControllerFactor	Reglerfaktor einstellen	!ctrf
SetControllerSteps	Regler-Schritte einstellen	!ctrs
SetControllerTimeout	Regler-Überwachungs-Timeout einstellen [ms]	!ctrt
SetControllerTWDelay	Reglerverzögerung setzen	!ctrd
SetCtrFastMoveOff	Fast Move Funktion „AUS“	!ctrfm 0
SetCtrFastMoveOn	Fast Move Funktion „EIN“	!ctrfm 1
SetTargetWindow	Reglerzielfenster einstellen	!twi

Trigger-Ausgang

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
GetTrigCount	Triggerzählerstand setzen	?trigcount
GetTrigger	Einstellung vom Trigger auslesen	?trig
GetTriggerPar	Trigger Parameter auslesen	?triga ?trigm ?trigs ?trigd
SetTrigCount	Triggerzählerstand lesen	!trigcount
SetTrigger	Trigger Ein/ Aus	!trig
SetTriggerPar	Trigger Parameter	!triga !trigm !trigs !trigd

Snapshot-Eingang

API-Befehl	Kurzbeschreibung	LSTEP-Befehl
GetSnapshot	Einstellung vom Snapshot auslesen	?sns
GetSnapshotCount	Snapshot-Zähler	?snsc
GetSnapshotFilter	Eingangsfilter auslesen	?snsf
GetSnapshotPar	Snapshot-Parameter auslesen	?snsl ?snsm ?sns
GetSnapshotPos	Snapshot-Position auslesen	?snsp
GetSnapshotPosArray	Snapshot-Position aus Array auslesen	?snsa
SetSnapshot	Snapshot Ein/Aus	!sns
SetSnapshotFilter	Eingangsfilter setzen	!snsf
SetSnapshotPar	Snapshot-Parameter	!snsl !snsm !sns